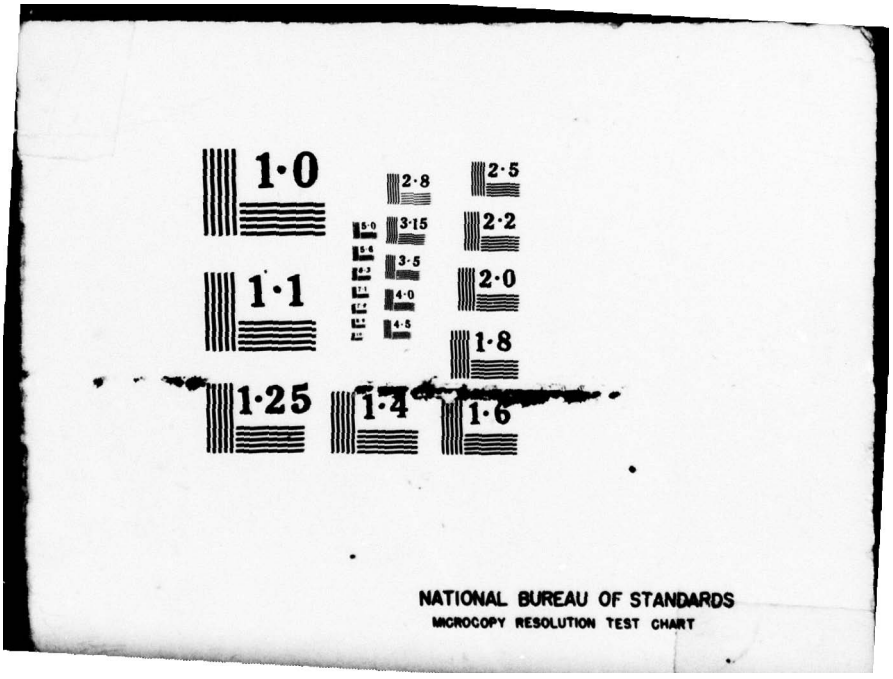
AD-A062 147 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO F/G 9/2
ENHANCEMENT AND EXTENSIONS TO THE XPL COMPILER GENERATOR SYSTEM--ETC(U)
AUG 78 G R FELDBAUER

UNCLASSIFIED AFIT-CI-79-85 NL

1 OF 2
AD
A062 147


1·0
1·1
1·25
2·8
3·15
3·5
4·0
4·5
1·4
2·5
2·2
2·0
1·8
1·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: CI 79-85

2. GOVT ACCESSION NO.: LEVEL II

3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): Enhancement and Extensions to the XPL Compiler Generator System

5. TYPE OF REPORT & PERIOD COVERED: Thesis

6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): Captain Gary R. Feldbauer

8. CONTRACT OR GRANT NUMBER(s): AFIT-CI-79-85

9. PERFORMING ORGANIZATION NAME AND ADDRESS: AFIT Student at West Virginia University

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:

11. CONTROLLING OFFICE NAME AND ADDRESS: AFIT/CI, WPAFB OH 45433

12. REPORT DATE: Aug[illegible]78

13. NUMBER OF PAGES: 91 (178p.)

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Master's thesis

15. SECURITY CLASS. (of this report): UN CLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release, Distribution Unlimited

DDC RELEASED DEC 13 1978 F

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

18. SUPPLEMENTARY NOTES

DEC 4 1978

JOSEPH P. HIPPS, Major, USAF
Director of Information, AFTT

APPROVED FOR PUBLIC RELEASE AFR 190-17.

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

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

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

012 200

AD A062147

DDC FILE COPY

79-85

Enhancement and Extensions to the XPL Compiler Generator System

by

Gary R. Feldbauer

Captain,

United States Air Force

Submitted to

The Department of Statistics and Computer Science

of

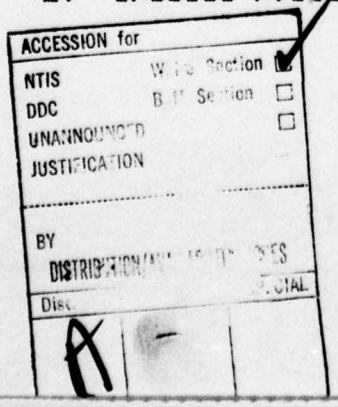
West Virginia University

In Partial Fulfillment of the Degree of

Master of Science in Computer Science

August 1978

Table of Contents



List of Figures

I. INTRODUCTION

A. History

The compiler generator system, known as XPL, was developed during the mid to late 1960's as an aid to students of compiler design and individuals interested in creating formal language translators or computer programming language compilers [1]. The present XPL system consists of several independent program modules: a grammar analysis program (ANALYZER), used to convert a formal language specified in Backus-Naur Form (BNF) into acceptable syntax tables for the XPL compiler; the XPL compiler (XCOM), a single pass table-driven syntax directed translator which creates executable machine language instructions for the IBM 360/370 series of computer systems; an ASSEMBLY language submonitor (XPLSM), designed to load the compiled object instructions into the main computer memory and provide auxiliary services such as input/output interface to the operating system; a character-string compression module (COMPACTIFY), a run-time subroutine, used to rearrange character strings into contiguous memory locations and reallocate unused memory space for future character-string manipulations; and, finally, a prototype compiler (SKELETON), which may be used as a foundation for the creation of a user tailored compiler.

All modules which comprise the XPL system are written in the language XPL, with the exception of the submonitor as was previously noted. The language XPL is a PL/I type language designed specifically for the purpose of creating compilers. Even though the language does not contain all the facilities of the

PL/I language, there are added features not available in PL/I to facilitate the creation of language compilers.

The theory of language design and the XPL system is discussed in great detail in the text "A COMPILER GENERATOR". It is strongly suggested that the reader of this report refer to the XPL text, in particular chapters six thru ten, to gain a greater appreciation of the XPL language and obtain a better understanding of the theory and practices outlined in this report.

B. Limitations

The XPL system is a very efficient and useable design and implementation aid. However, the compiler imposes several restrictions and lacks certain facilities which limit the overall flexability of the compiler generator system.

The restrictions which should be evaluated are:

1. The "SUBSTR" function is not allowed to be used as the target of an assignment.

2. Negative values may not be used as step variables in DO-loops.

3. Character-string variables may share descriptors and inadvertent modification of shared strings is possible.

4. Predefinition of reserved word abbreviations is not automatically supplied (i.e. DCL is not an abbreviation for DECLARE).

The added facilities which should become a part of the XPL system are:

1. External modules and an external module library, to eliminate the need to compile built-in functions and run-time routines along with the compilation of a user module.

2. An external data type (such as PL/I EXTERNAL and FORTRAN LABELED COMMON).

3. PL/I type data structures and arrays of structures.

C. Problem

The major function of this problem report is to attempt to eliminate most of the limitations stated in the previous section and produce a new version of the XPL compiler generator system which is more versatile and even a more valuable tool to persons interested in language design and compiler construction.

The specific objectives of this problem report are as follows:

1. Creation of IBM linkage editable object modules.

2. Provide a vehicle for external program modules and data variables.

3. Creation of a subroutine library of XPL system modules.

4. Provide a solution to the character-string assignment anomaly.

D. Solution

After a complete and exhaustive examination of the XPL compiler XCOM, it was determined that the objectives of this project report could be attained.

Creation of the ESD, TXT, RLD, and END records required by the IBM LINKAGE EDITOR is done after all object code is produced by XCOM. This minor second pass of the compiler is done to ensure the present code generation technique is not modified, thereby reducing the possibility of creating erroneous object code.

External program modules and data variables are implemented through the use of a pseudo syntax in conjunction with the label processing software of the XCOM compiler. The mechanism is triggered by the identification of the special purpose labels 'COMMON:', 'EXTERNAL:', and 'PUBLIC:'.

The extended data variable processing is invoked when the pseudo label 'COMMON:' is detected preceding a DECLARE statement. When a data variable assumes the attribute of common, it becomes global to all modules in which it is declared as common and follows the rules governing 'Scope of Variables' within the XPL language.

The label 'EXTERNAL:', preceding a procedure definition, will identify the entry point and formal parameter list of a procedure to be referenced within the present module, but has been compiled previously and is assumed to be resident in the XPL system library or a user module library.

If an XPL procedure is preceded by the label "PUBLIC:", it is compiled as a global entry which may be invoked by any module which contains an external definition of the procedure name.

Figures 1.1 and 1.2 contain two program segments which exemplify the use of the pseudo labels common, external, and public.

The string assignment problem is solved by forcing a new copy of a character-string within the assignment portion of the XPL "SYNTHESIZE" routine. Since this modification is also required when "COMMON" character-strings are referenced, the additional XPL statements to include all character-strings is minimal.

These modifications will provide a number of advantages to the user of the XPL compiler generator system. The theory behind the external and public attributes will allow the capacity of "Plug Compatible" software modules within the programming system. If an individual is not completely satisfied with the performance of a module (i.e. SCANNER, RECOGNIZER, ERROR routines, etc...), it will be possible to redesign only the routine in question without the need to recompile all modules within the system.

The use of "COMMON" data variables will allow inter-module communication and may be used to reduce the overhead involved in parameter passing.

The character-string modification, although requiring added overhead at program execution time, will eliminate the annoying possibility of modifying several strings when only one was intended to be changed.

```
/* EXAMPLE OF COMMON DATA VARIABLES AND
   THE DEFINITION AND USE OF AN EXTERNAL PROCEDURE        */

  DECLARE (I,J,K) FIXED; /* INTERNAL DEFINITION  */
  DECLARE Z       BIT(1);
  COMMON: DECLARE  A  BIT(1);  /* "COMMON" DECLARATION   */
  COMMON: DECLARE  B  FIXED,
                   (C,D,E)  BIT(16),
                   F(10)    CHARACTER;

 /* THE FOLLOWING PROCEDURE IS AN EXAMPLE OF AN EXTERNAL
    DEFINITION OF AN ENTRY POINT  ("REFERENCE DEFINITION").  */
  EXTERNAL: PGMA: PROCEDURE (X,Y,Z) FIXED;
            DECLARE (X,Y,Z) FIXED;
            END PGMA;

                      .
                      .
                      .
   I = J + PGMA(Z,B,F(K)) - K;
/* THE ABOVE LINE OF CODE IS AN EXAMPLE OF AN
   "EXTERNAL" FUNCTION REFERENCE                          */
                      .
                      .
                      .
  EOF    /* END OF THE EXAMPLE MODULE */
```

Figure 1.1 XPL Program Example using COMMON and EXTERNAL

```
/* EXAMPLE OF COMMON DATA VARIABLES AND
   THE DEFINITION AND USE OF A 'PUBLIC' PROCEDURE   */

DECLARE X   FIXED;   /* INTERNAL DECLARATION  */
COMMON:  DECLARE   B   FIXED;   /* 'COMMON' DECLARATION */
COMMON:  DECLARE   D   BIT(16),
                   F(5) CHARACTER;

PUBLIC:  PGMA:  PROCEDURE (P1,P2,P3) FIXED;
         DECLARE P1   BIT(1),
                 P2   FIXED,
                 P3   CHARACTER;
                      .
                      .
                      .
        IF P1 THEN
           X = LENGTH(F(P2)) / B - D;
        ELSE
           X = LENGTH(F(P2-1)) * B + D;
                      .
                      .
                      .
        RETURN (X - LENGTH(P3) + 2);
                      .
                      .
                      .
        END PGMA;
                      .
                      .
                      .
 EOF   /* END OF EXAMPLE PROGRAM */
```

Figure 1.2 XPL Program Example using COMMON and PUBLIC

E. Programming Practices and Project Restrictions

Two important assumptions were made prior to initiating this project. One concerns the underlying philosophy that the XPL system, except for the submonitor, is written completely in the language XPL. This philosophy has been maintained and all modifications to XPL modules are implemented using the XPL language. The second assumption, which also concerns the XPL language, is that no modification to the XPL semantic grammar would be made. Since a very good discussion of the XPL language is contained in the text "A COMPILER GENERATOR", it was felt the XPL language should remain as described in the text, and this report used as a supplementary document for individuals interested in developing compilers using the extended capabilities outlined herein.

II. Pseudo Syntax to Provide Extended Capabilities

The implementation of version IV of the XPL compiler XCOM, to provide the extended capability of external program modules and common data variables poses three distinct requirements:

1. How will the user identify to XCOM, given that the XPL grammar will not be modified, that an extended capability is to be invoked.

2. What is required by the IBM Linkage Editor to identify and process variables with extended attributes.

3. What additional processing must be performed by the XPL compiler when a variable with extended capability is defined and referenced.

This chapter deals with a discussion of each of these requirements.

A. Syntax Identification

All labels within the XPL language are defined by the single BNF production "<LABEL DEFINITION> ::= <IDENTIFIER> : " [1], which means that when an identifier is followed by a colon, the identifier assumes the attribute of a label. This rule provides the major "hook" into the XPL version III grammar to provide the facility for external program modules and common data variables.

Through a minor modification to the label processing portion of the XCOM procedure "SYNTHESIZE" and use of the global variable "ESDTYPE", the identification of external modules and common data

variables is accomplished. The processing of labels in version IV of XCOM proceedes in the following manner:

1. If the length of an identifier is greater than 8 characters, the identifier is entered into the symbol table as a label and no further processing is performed (the length of the longest pseudo variable is 8 characters).

2. If the identifier is 'PUBLIC', 'EXTERNAL', or 'COMMON', the global variable ESDTYPE is set to 1,2,or 5 respectively, and the pseudo label is not entered into the symbol table.

3. If the identifier does not match any of the pseudo labels, the identifier is entered into the symbol table as a label and the variable ESDTYPE is not referenced.

Since all statements within the XPL language may be preceded by any number of labels, and the variable ESDTYPE is not reset until the XCOM parsing mechanism 'reduces' a statement into a statement list, the variable ESDTYPE will contain a value to identify if an extended option is to be invoked during the processing of the next XPL statement. This feature allows the implementation of the extended capabilites of the version IV xcom compiler.

B. Linkage Editor Requirements.

The version IV XCOM compiler uses the IBM linkage editor as the vehicle responsible for combining external object modules into a single load module and providing address resolution of the external and common variables. It is, therefore, necessary for XCOM to supply the Linkage Editor with the proper ESD information to facilitate production of correct and complete load modules [2].

The scalar "SYESDTYPE" has been added to the XCOM version III symbol table variables which will contain the Linkage Editor "ESD type". This variable is set to the value contained in ESDTYPE when an identifier is entered into the symbol table during the processing of a "Declaration Statement" or "Procedure Definition".

The variable SYESDTYPE serves two purposes; One being in the creation of the Linkage Editor ESD records, and the other, as a method of identifying to the XCOM synthesis procedures that additional processing must be performed. Both of these purposes will be discussed in greater detail later in this report.

C. Definition and Referencing Processing

The technique outlined above to invoke an extended capability and identify a variable which is to assume an extended attribute is a very minor portion of the processing required to provide the facility of external program modules and common data variables. The processing required, by XCOM, for definition and accessing of these variables is much more complicated and will be discussed here. The reader is encouraged to refer to figures 1.1 and 1.2 which contain the actual XPL statements used to identify the extended capabilities.

Definition of COMMON Variables

All data variables which are to assume the attribute "COMMON" must be defined in a "Declaration Statement" which was preceded by the pseudo label 'COMMON:'. This will cause the variable ESDTYPE to be set to "05" and be recognized within the XCOM storage allocation procedure "ALLOCATE". Allocation of storage for common variables proceeds as follows:

1. Force the data pointer (DP) of the next available storage location to a full word boundary.

2. Determine the amount of storage to be allocated depending on the type and dimension of the variable (i.e. FIXED data variables require 4 bytes of storage for each occurrence of the variable).

3. Reserve one word of storage to contain the relocated absolute address of the common variable.

4. Determine the address of the word reserved in base and displacement form and place these values into the symbol table as the address of the variable (SYBASE and SYDISP).

5. Set the symbol table variable "SYLEN" to the number of bytes to be allocated by the Linkage Editor for the common variable.

6. Set the symbol table variable "SYRLDADDR" to the data pointer of the word to contain the absolute address of the variable.

It should be noted that a variable with the common attribute requires only one word of storage in the defining module, for the absolute address of the variable. The actual storage for the common variable will be allocated by the Linkage Editor and Loader prior to execution of the user program. Also, a character-string variable will not require a descriptor value in the defining module's descriptor array. However, the common character-string will be allocated a descriptor array of its own (this will be discussed in the next section).

Definition of EXTERNAL Modules

An external module entry point and parameter list is identified when the pseudo label "EXTERNAL:" precedes the XPL definition of a procedure. The defining of a procedure in this manner can be called a "reference definition" because the purpose is to define to the XCOM compiler that a module which has been compiled externally will be referenced within the present module

via a call or function reference.

The definition of an external module requires special processing of the parameter list and the actual module call. The processing of the parameter list is as follows:

1. Enter the parameter identifier into the symbol table.

2. Allocate one word in the data area of the main module which will contain the value of the parameter referenced via call or function reference.

3. Set the symbol table variable SYESDTYPE to hexadecimal "0F" (note: a hexadecimal constant is defined as a numeric value enclosed in quotes. This notation will be used extensively in this report.) to inhibit further allocation processing during declaration of the variable. When the ALLOCATE procedure detects an "0F" in SYESDTYPE, the variable SYESDTYPE is set to zero and no other processing is performed.

The processing of the external module proceeds as follows:

1. After all initialization processing has been completed by the procedure PROC_START, the procedure "BUILD_EXTERNAL_CALL" is called. The following steps are performed by the procedure BUILD_EXTERNAL_CALL.

2. Allocate a data word to contain the present value of the "string base register" (register 13).

3. Store the contents of register 13 into the data word.

4. Allocate 18 data words to be used as the linkage "save area".

5. Load register 13 with the address of the save area.

6. Allocate a data word which will contain the absolute relocated address of the external entry point.

7. Store the data pointer to the above word in the symbol table variable SYRLDADDR which corresponds to the procedure name (PROCMARK - 1).

8. Load register 15 (the entry point register) with the value contained in the word allocated in step 6.

9. Load register zero with the address of the first parameter. (note: This address points to the word which contains the value of the parameter, not the address.)

10. Emit a Branch and Link instruction with register 15 as the entry point register and register 14 as the return register.

11. Load register 13 with the string base address saved in step 3.

12. Emit the code for an uncondition return from the "reference definition" module to the instruction following the call or function reference.

Definition of PUBLIC Procedures

A procedure which is to assume the attribute of PUBLIC or "defining definition" is identified by the pseudo label "PUBLIC:" preceding a procedure definition and the variable ESDTYPE set to "01". This will trigger XCOM processing which will save all of the callers registers into the callers save area, load the relocated working registers for the module, and store the address of the parameter list for later use.

The definition of the parameter list identifiers is processed in the following manner:

1. Enter the parameter identifier into the symbol table.

2. Set the symbol table variable SYESDTYPE to "0E" to indicate to the ALLOCATE procedure that a public procedure parameter has been declared and a parameter value must be loaded into the location of the variable.

After all initialization processing has been accomplished by the PROC_START procedure, the "BUILD_PROLOGUE" procedure is called to complete the entry initialization as outlined below:

1. Set the program pointer (PP) to a full word boundry.

2. Set the symbol table variables SYBASE and SYDISP to the address of the procedure (PP from step 1).

3. Emit code to branch around the public procedure name and address of the data area for the module, using register 15 as the program base register.

4. Store the entry point name, padded to 7 characters, and preceded by its length into the program area.

5. Set the symbol table variable SYRLDADDR to the current value of the program pointer (PP), then increment the PP four bytes (the relocated address of the data CSECT will be placed here by the Linkage Editor and Loader).

6. Store the callers registers into the save area addressed by register 13.

7. Load register 11, the data base register, with the value contained at the address allocated in step 5, using register 15 as the program base register.

8. Load the relocated data and program base registers for this module from the data area, using register 11 as the base register.

9. Store the save area address register (register 13) into this module's save area (bytes 4-7).

10. Load and store the address of this module's save area into the caller's save area (bytes 8-11).

11. Load registers 13 thru 15 from this module's register relocation area (BASEDATA + 52).

12. Reserve one data word for the parameter address.

13. Set the variable PARMLOC to the data pointer of the parameter address save area.

If the declarations for the parameters immediately follow the procedure definition, the code to load the parameter values from the parameter area into the data area will appear following the procedure initialization code.

Referencing COMMON Variables

The technique used by the XPL language to manipulate data variables is to load the data into an accumulator register (register 1-3), perform the required processing, and, if the variable appears on the left of an assignment, store the value from an accumulator register into the data area. The symbol table variables SYBASE and SYDISP contain the address of the data variable in base/displacement form. If the variable does not have an ESDTYPE of '05', the data may be accessed directly from the data area. However, when the variable has the attribute COMMON (SYESDTYPE = '05'), the address of the data must first be loaded into a scratch register (register 14) from the data area specified by SYBASE and SYDISP. The actual data is then loaded or stored, using the scratch register as the base register for the variable.

Indexing of arrays does not present any special processing requirements because all accessing to and from the data area is done using IBM 'Register to Indexed Storage' (RX) type instructions and all array subscripts are contained in an index register. If the variable is not subscripted or the subscript of zero is specified, the index register is cleared to zero.

The referencing of common character-strings presents a special requirement because of the way character-strings are

processed by the XPL compiler, in general, and the specific implementation of character-strings in version IV of XCOM. Before discussing the special processing required for character-string manipulation in version IV, an overview of the general character-string processing will be presented.

Implementation of character-strings is done using character-string descriptors and a loosely allocated area of computer memory for the storage of the character-strings. The descriptors are located in a fixed contiguous area of memory addressed by the "string base register" (register 13) and contain the current address and length of a particular character-string. The address portion of the descriptor is the absolute address where the string is located in the "free string area". Also, if the length and address are zero, the string is considered to be a null or unallocated string.

Since version III of XCOM does not support external program modules or common data variables, there is only one descriptor list for all character-strings and one free string area. When a string is accessed, the symbol table variables SYBASE and SYDISP contain the address of the string descriptor, which is loaded into an accumulator, processed, and a new descriptor value may replace a previous descriptor if the operation of assignment or concatination are performed.

The free string area in memory is compacted from time to time for the purpose of reallocating previously used and now unused portions of memory, to allow further string operations which require space in the free string area. The version III

compactification procedure 'COMPACTIFY' is designed to process only one string area; This procedure required considerable modification for the version IV implentation of the XPL system, which allows any number of string areas.

Version IV of the XPL compiler, XCOM, will provide the capability to access character-strings in a number of common or main module string areas. The specific implementation requires that each string area contain its own string descriptor list and free string area. The descriptor list contains all of the information pertaining to the string area. This information is contained in a 5 word (20 byte) descriptor preamble which is located at the start of each string descriptor list and uses 5 string descriptors. Figure 2.1 defines the contents of the descriptor preamble.

Character-strings are referenced in two ways in the version IV XPL compiler, depending on whether or not the character-string is defined as a common variable.

When a character-string is referenced, and has the attribute COMMON, the string descriptor is loaded from the descriptor area for that string in a similar manner as was discussed above concerning arithmetic variables. One additional step must be performed when accessing a common string descriptor; After the address of the string descriptor is loaded into the scratch register (register 14), the area allocated to the string preamble must be added to ensure that the correct descriptor will be accessed. This is done by inserting an offset of 20 bytes into the displacement portion of the RX type load or store instruction

WORD/BYTE	CONTENTS
0 / 0-3	The 'LOWER_BOUND' of the free string area used to optimize the procedure XPLPACK
1 / 4-7	Absolute address of the next available byte in the free string area (FREEPOINT).
2 / 8-11	Absolute address of the first available byte in the free string area after the string descriptors and constant character-strings are allocated (FREEBASE).
3 / 12-15	Absolute address of the last byte available in the free string area (FREELIMIT).
4 / 16-19	The number of string descriptors allocated to this string area.

Figure 2.1 String Descriptor Preamble

used to access the common string descriptor. Since this offset is already accounted for when establishing the descriptor base for strings which are part of the main module, it is not required to add the 20-byte offset when accessing string descriptors without the common attribute.

Referencing External Modules

The actual invoking of an external module via a call statement or function reference is quite simple within the version IV implementation of external modules. It is simple because no special processing is required for the parameter passing or module linkage. All special processing required to access the external module is contained within the external "reference module", so when an external module is invoked, the input parameters are "stuffed" into parameter locations defined in the "reference module" and an actual call (branch and link) is executed which causes the "reference module" to be entered.

It is within the "reference module", as was discussed in the section concerning the definition of an external module, where all special processing occurs. Therefore, an external entry may be invoked at many different locations within a program, but the code required to provide the parameter and call linkage appears only once within the "external reference" module.

III. Creation of the Linkage Editor file 'SYSLIN'

As was mentioned in the introduction to this report, a major objective of the project is to create an object deck which is acceptable as input to the IBM Linkage Editor. Along with producing the linkage editor TXT records from the machine language instructions produced by the XCOM compiler, it is necessary to produce ESD records to identify the CSECTs (control sections) for the main object program and external variables. Also, RLD records are produced to ensure proper addressing of register values, string descriptors, and addresses of externally referenced modules and common data variables.

Chapter II contained an explicit discussion of how the external modules and common data variables are identified and accessed. This chapter will explain how the linkage editor records are produced from the information provided by the XPL compiler.

In general, the Linkage Editor object input file, 'SYSLIN' is produced after all statements in the user's program have been compiled without severe compilation errors.

If severe errors are detected during the compilation process, a message is written to the list file 'SYSPRINT' indicating that severe errors have been detected and the creation of the SYSLIN file will not be attempted. If no severe error were encountered, the Linkage Editor records are produced in the following order: ESD, TXT, RLD, and END, for the main program CSECTs. When the compiler detectes that a common character-string has been

declared, a separate set of ESD, TXT, RLD, and END records are produced for each common character-string defined.

The Linkage Editor records are written to the data set 'SYSPUNCH' which is explicitly defined as the XCOM file 'OUTPUT(2)' [1]. Each record contains 80 bytes of information and the records are blocked 10 records per block. The DD statements used by the compiler and link steps of the XPL JCL procedures (see appendix A1 - A3) are as follows:

```
//SYSPUNCH DD DSN=&&LOADSET,DISP=(MOD,PASS),UNIT=SYSDA,
//    SPACE=(800,(200,100)),DCB=(RECFM=FB,BLKSIZE=800,LRECL=80)

//SYSLIN  DD DSN=&&LOADSET,DISP=(OLD,DELETE)
```

A. ESD Records.

Four types of ESD records are produced by the XCOM procedure 'PUNCH_ESD'. The SD (section definition) records identify the three major CSECTs of any module compiled (program, data, and strings). The LD (label definition) records identify module entry points which have the attribute PUBLIC. The ER (external reference) records identify module entry points which have the attribute EXTERNAL. The CM (common) records identify the data variables which have the attribute COMMON.

Each ESD sub-record requires 16 bytes of information and these records are packed three sub-records per 80 byte SYSLIN record. The format of the ESD sub-record is contained in figure 3.1.

ESD Record Format:

Position	Contents
1	'02'
2-4	ESD
5-10	Space
11-12	Byte Count - number of bytes of data in positions 17-72
13-14	Space
15-16	ESD-id of the first non-LD type entry, space if all entries are LD
17-72	ESD entries (See below)
73-80	Space or desck-id and sequence number

ESD Entry Format:

Relative Position	Contents
1-8	Name of ESD entry padded to 8 characters
9	ESD type '00' = SD, '01' = LD, '02' = ER, '05' = CM
10-12	Relative address of an SD or LD type ESD entry
13	Space
14-16	Length of control section for ESD types '00' and '05', ESD-id of Program CSECT for ESD types '01', or Space for '02' ESD types

Figure 3.1 ESD Record Format

The three SD records contain the name of the program, data, and string CSECTs which is taken from the first "PUBLIC" definition of an entry or the fixed name "XPLMAIN". If the public name is used and is less than 7 characters, the name is padded to a length of 7 characters with all blanks replaced by asterisks, and the number 1,2 or 3 is appended. For example, if the first definition of a public entry was the name "PGMA", then the CSECTs for the program, data, and string areas would be identified as "PGMA***1", "PGMA***2", and "PGMA***3". This is done to differentiate the modules when the Linkage Editor "MAP" and "XREF" options are invoked.

After the three ESD sub-records have been produced for the above CSECTs, the global variable "ESDID" is set to four, the ESD-identifier of the next non-label ESD item to be produced.

The information to produce the LD, ER, and CM ESD sub-records is contained in the symbol table variables SYT (name), SYESDTYPE (type), SYLEN (length), and the pair SYBASE and SYDISP (address). Therefore, to create the ESD entries for a variable with extended capability, it is necessary to perform a serial search of the symbol table looking for non-zero values in the variable SYESDTYPE. When this variable does contain an ESD type identifier, the symbol table pointer to this entry is placed into the array "CASESTACK" and saved for processing of the RLD records. This is done to prevent a second serial search of the symbol table. After the symbol table pointer is saved, the entry is processed in one of the three methods described below.

If the variable SYESDTYPE contains the value "01" (LD or PUBLIC definition of an entry point), an ESD sub-record is created containing the following information from the symbol table entry: name, ESD-type, address of the identifier, and the ESD-id of the program CSECT. The global variable ESDID will not be incremented for this label definition entry because the Linkage Editor does not recognize ESD-id's for LD type entries.

If the variable SYESDTYPE contain the value "02" (ER or EXTERNAL definition of an entry point), the ESD sub-record will be created with the name, ESD-type, a zero value for the address, and blanks in the fourth field. The global variable ESDID is inserted into the symbol table variable "SYTCO" which will be referenced during the creation of RLD records. The variable ESDTYPE will then be incremented.

A SYESDTYPE of "05" identifies a CM Linkage Editor ESD-type for a variable with the attribute COMMON. The ESD sub-record will contain the following information: name, ESD-type, zero, and the length of the variable from SYLEN. As was done for an ER sub-record, the ESDID is stored into the variable SYTCO and then incremented. Also, if the variable is a character-string common variable, the symbol table pointer is saved in the array "FIXCADR" to be used during the creation of the special CSECT for common character-strings.

To simplify the creation of ESD records, a specialized procedure was developed to enter each ESD sub-record into the 80 byte ESD output record.

When each ESD sub-record is to be created, the procedure 'OUTPUT_ESD' is called and receives the four portions of information required for each ESD sub-record. The OUTPUT_ESD procedure will ensure that the sub-record is placed into the array 'BINARY' in the correct positions and that when three sub-records have been created, the ESD record is written to the SYSPUNCH data set.

After all symbol table entries have been examined and all ESD sub-records have been created a final check is made to ensure any partial records in the array BINARY are written to SYSPUNCH.

B. TXT Records

The Linkage Editor TXT records for the program, data, and character-string CSECTs are produced by the XCOM procedure 'PUNCH_TXT'. The records are produced in ESD-id order starting with the program CSECT. The format of the TXT record is contained in figure 3.2.

The procedure PUNCH_TXT is generalized to build the three CSECTs from the three XCOM work files 'FILE1', 'FILE2', and 'FILE3', which contain the machine language instructions (program), data area (data), and character-string constants (strings). The only exception is that the character-string descriptor array 'DESC' is produced prior to the creation of TXT records for the constant strings. The method of producing each TXT CSECT can be summarized as follows:

1. Initialize the variable 'BASEADDR' to the address of the binary array 'CODE'. BASEADDR will contain the absolute

TXT Record Format:

Position	Contents
1	'02'
2-4	TXT
5	Space
6-8	Origin addres of this TXT record
9-10	Space
11-12	Byte count - number of bytes of data in positions 17-72
13-14	Space
15-16	ESD-id for this text CSECT
17-72	Text for Program, Data, or String CSECT
73-80	Space or deck-id and sequence number

Figure 3.2 TXT Record Format

memory address of the first byte of information to be moved to the TXT work area in the array BINARY.

2. Initialize the variable 'TXT_LIM' to reflect the maximum number of bytes which can be moved from the array CODE before the next block of information must be read from one of the work files.

3. Set the variables 'TXT_ORG' and 'TXT_END' to the relative bounds of the TXT CSECT being created.

4. Set the variables ESDID and 'FILE_ID' to their respective values (e.g. program = 1).

5. Set the origin address and ESD-id into the array BINARY.

6. Repeat steps 8 thru 11 until all bytes of text have been transferred to a TXT record.

7. If any TXT bytes remain in the array BINARY, write the partial TXT record to the SYSPUNCH data set.

8. If the address of the current byte to be moved from the CODE array to the array BINARY is equal to the value contained in the variable TXT_LIM, then read in the next block of data from the work file corresponding to the value in FILE_ID, reset the variable BASEADDR to point to the start of the CODE array, and update the variable TXT_LIM to reflect a new limit value.

9. If the limit has been reached for the array BINARY, then write the TXT record to the data set SYSPUNCH and

update the area in the array BINARY to reflect the current origin address of the next TXT record to be produced.

10. Move one byte of text from the CODE array to the array BINARY using the 'COREBYTE' function as addressed by the variable BASEADDR ('BINARY(BINPTR) = COREBYTE(BASEADDR);').

11. Increment the subscript for the array BINARY ('BINPTR') and the address of the next byte in the array CODE (BASEADDR).

A minor modification to the above procedure is required when creating the character-string CSECT (CSECT 3).

During the compilation of all XPL programs by XCOM, character-string constants are processed by allocating a word in the descriptor array 'DESC' which contains the length and relative offset into the constant string work area. Since both descriptors and character-strings are contained within the same CSECT and the method of string compaction requires that all string descriptors immediately precede the constant strings in contiguous memory locations, it is necessary to create the character-string CSECT by building TXT records containing the descriptors followed by the character-string constants. The descriptor TXT records are created by setting the variable BASEADDR to the base address of the array DESC, and do not read information from a work file, but instead, move data from the descriptor array into the array BINARY. This was done to preclude modification to the DO-loop which moves the data from a location addressed by the variable BASEADDR into the array BINARY.

When the character-string constant TXT records are created, the relative origin address is set to the end of the descriptors which were allocated during the present compilation. The reader is encouraged to reference Appendix 6 for the actual XPL statements required to build these records.

C. RLD Records

The Linkage Editor RLD records are generated for three purposes: Relocation of the base address registers for each program module, relocation of character-string descriptors, and, the relocation of external entry point addresses and the base address of each common data variable.

The XCOM procedure "PUNCH_RLD" was included in the version IV XPL system for the specific purpose of creating the RLD records. Similar to the sub-procedure used to create ESD records, the sub-procedure "OUTPUT_RLD" provides the same services. The format of the Linkage Editor RLD records can be found in figure 3.3.

The process of relocation, pertaining to the version IV XPL system, is performed to ensure that all relative address constants (ADCONs) and addresses of external variables (VCONs), which are referenced during the execution of an XPL program, but are unknown during compilation, are resolved to absolute memory addresses. The actual relocation process is performed by the IBM Linkage Editor and Loader, based on the RLD records produced by XCOM, prior to and during the "loading" of the XPL program into computer memory.

RLD Record Format:

Position	Contents
1	'02'
2-4	RLD
5-10	Space
11-12	Byte Count - number of bytes of data in positions 17-72
13-16	Space
17-72	RLD Entries (See below)
73-80	Space or deck-id and sequence #

RLD Entry Format:

Position	Contents
1-2	Relocation Pointer - ESD-id of LD, ER, or CM ESD entry
3-4	Position Pointer - ESD-id of SD ESD entry
5	Flag Byte (See below)
6-8	Relative address within the 'Position Pointer' CSECT

Flag Byte Format:

Position (Bit)	Contents
0-3	Flag type - 0000 - non branch (ADCON) 0001 - branch (VCON)
4-5	Length Specifier - 10 - 3 bytes 11 - 4 bytes
6	Direction bit - 0 - positive 1 - negative
7	Type of next RLD entry - 0 - different R and P ptr's 1 - same R and P pointers

Figure 3.3 RLD Record Format

The information needed by the Linkage Editor and Loader, and provided by XCOM, to relocate any ADCON or VCON is:

1. A relocation pointer (R_PTR), which is the ESD-id of the CSECT whose address is unknown at compilation (VCON), or must be added to a known relative location (ADCON).

2. A position pointer (P_PTR), which is the ESD-id of a CSECT which contains the ADCON or VCON.

3. The relative location (offset) of the ADCON or VCON within the CSECT identified by the P_PTR. This is the area which will be modified by the Linkage Editor and Loader and will receive the absolute address of the CSECT identified by the R_PTR.

4. A "FLAG" byte to indicate to the Linkage Editor the number of bytes to be modified, the type of relocation, the direction of relocation, and if the R_PTR and P_PTRs from a previous RLD sub-record are to be reused to relocate a given offset.

Within every XPL program, 16 words in the data area are reserved to contain the relocated register values (ADCONs) which are loaded at the beginning of each PUBLIC or main entry point of the module. Relocation of the data base registers (registers 4-11), the program base register (register 12), the string base register (register 13), the address of the entries XPLPACK and XPLMON (register 14 and 15) is performed through the use of RLD records.

The absolute address of the character-string CSECT is added to all non-zero descriptor locations (including the preamble) to ensure addressability of character-string constants and the variables in the preamble required by the procedure XPLPACK. This is a major modification to the version III XCOM, which relocated all string descriptors through the use of a run-time built-in procedure after initial program load. This relocation routine, which was only executed once at each module initialization, has been eliminated.

The absolute addresses of all externally defined entry points and common data variables, are relocated as VCONs within the data CSECT.

The absolute address of the data CSECT is relocated into the program CSECT for each entry point defined as PUBLIC and at the beginning of each program module. This is done so base registers relocated into the fixed locations of the data CSECT may be loaded to establish overall addressability of the program module.

D. The END Record

When all of the ESD, TXT, and RLD records have been created, the final END record is written to the data set SYSPUNCH. This record contains the ESD-id of the program CSECT and the relative address of the initial entry, which is zero (see figure 3.4).

END Record Format:

Position	Contents
1	'02'
2-4	END
5	Space
6-8	Relative entry address
9-14	Space
15	ESD-id of the Program CSECT
16-80	Space

Figure 3.4 END record Format

E. Special Processing of Common Character-strings

The general nature in which all character-strings are processed by the XPL version IV string compaction routine XPLPACK and the processing of null character-strings requires that each common character-string area to be created as a separate Linkage Editor CSECT. Therefore, each common character-string, either a array or single character-string, must have its own set of ESD, TXT, RLD, and END records.

The ESD record identifies the common character-string variable as a CSECT (SD type) to the Linkage Editor.

The TXT records initialize the preamble and string descriptor values. The preamble values for FREEPOINT, FREEBASE, FREELIMIT, and the number of descriptors are created in their relative address form. Also, all actual string descriptor locations are initialized to zero to indicate that they are null string descriptors.

RLD records are created to relocate the preamble addresses as required by the XPLPACK procedure.

The END record terminates the set and performs no other function.

IV. Summary of Modifications to the Version III XPL System

The previous chapters have attempted to explain the two major changes to the version III XPL compiler generator system. The capability of external program modules and common data variables was discussed in chapter II, and the creation of the records necessary to produce a linkage editable object deck was outlined in chapter III. The purpose of this chapter is to review the additional modifications to the version III XPL system.

A. Register Assignment

Figure 4.1 compares the register assignment conventions used by both the version III and version IV XCOM compilers.

The version IV convention is used in an attempt to bring the XPL system into closer unity with the IBM convention of using register 15 as the entry point register, register 14 as the return register, and register 13 as the save area register [3].

The register convention which makes version IV still different from the accepted IBM convention is noted by XCOM's use of register 0 as the parameter address register versus the IBM standard of register 1. Also, XCOM continues to use register 3 as the function value return register, unlike IBM. However, XCOM has implemented, in version IV, the IBM policy of returning the "return code" to the operating system in register 15.

REGISTER	USAGE	
	VERSION III	VERSION IV
0	Scratch	Scratch
1-3	Accumulators	Accumulators
4-11	Data Base	Data Base
12	Return	Program Base
13	String Base	String Base and Save Area
14	Program Base	Return
15	Submonitor entry point	Program Linkage

Figure 4.1 Register Assignment Comparison

Since both XCOM version III and IV pass parameters "by value" and not "by address", it is felt the use of register 0 as the parameter address register is acceptable and will allow future changes to the XCOM compiler to use register 1 for addresses, thereby ensuring downward and lateral compatibility.

B. Calling and Linkage Conventions

The version III XCOM compiler, by not providing the facility for external program modules, does not require the complicated linkage technique required in the version IV compiler.

The calling convention, except for external program modules, is basically the same between both versions. However, the linkage convention between EXTERNAL and PUBLIC modules is completely new.

Each definition of an EXTERNAL module requires a corresponding definition of an entry defined as PUBLIC with an identical name [4]. The linkage between the entries follows the IBM calling convention very closely. The calling module (defined as EXTERNAL) must provide an 18 word linkage save area with register 13 containing the address of this area. When the external module (defined as PUBLIC) is invoked, the callers registers are saved in the save area, and its own registers are loaded from the "static" data area as part of the entry initialization [5]. Prior to returning to the calling module, all of the caller's registers are restored.

C. Accessing Common data

The technique for defining and referencing data variables which possess the attribute COMMON was discussed in detail in Chapter II. However, since this capability is a major enhancement in the version IV XPL compiler, the method deserves summary here.

All common data variables are contained within their own control section (CSECT). If the variable is not a character-string, the Linkage Editor will provide the storage for the variable, via processing of the COMMON (CM) ESD entries, and if a character-string is declared with the attribute COMMON, XCOM will generate a preinitialized CSECT for the variable. The address of the external CSECT will be relocated into the data CSECT and referenced by the symbol table variables SYBASE and SYDISP. The accessing of a common data item requires two RX-type load or store instructions; the first to load the absolute address of the variable into a scratch register, and the second to perform the operations of load or store.

One caution must be made concerning common data variables declared as numeric (BIT and FIXED), and that is: any numeric common variable declared will be uninitialized at program execution. This is very important, because XCOM will preinitialize all internal data variables to binary zero at compilation. The Linkage Editor will not provide this service, and users should be wary of initial values in a area declared COMMON.

D. Character-string Allocation, Manipulation, and Compaction

The methods employed to allocate, modify, and compress character-strings , both internal and common have been redesigned significantly in the version IV XCOM compiler. This section will identify each of the changes in detail.

Character-string Allocation

The allocation policy of the version IV XPL compiler attempts to provide each character-string CSECT with sufficient free string area to both conserve main computer storage and ensure that unnecessary string compaction will not occur.

There are two bases for string allocation: If the character-string being defined is part of the main program segment (not common), the free string area will be alloted 192 bytes for each string declared. If the character-string is defined as COMMON, on the other hand, the free string area will receive 128 bytes of memory for each occurrence of the string.

The reason for this allocation policy is that all character-string manipulation, which requires intermediate results (e.g. concatination), will be performed using string space in the main program segment. Therefore, the possibility of time consuming string compaction is greater within the string area alloted to the main segment. Also, the allocating of a larger area in the main segment will reduce compaction time because the collection of unused string space will be performed using a "minor collection" (partial) algorithm rather than the alternative "major collection" (complete string reorganization) algorithm.

Character-string Manipulation

The major difference between the version III and version IV XCOM compilers, in the area of string manipulation, concerns the method of string assignment.

The technique implemented in version III is to assign string descriptors to variables and allow sharing of the data in the string free space. This method is very efficient during the execution of the XPL program (4 bytes of core are manipulated versus the actual number of bytes in the string). However, the sharing of a single string space does cause one very serious problem to the uninitiated user of the XPL compiler. This is the string assignment anomaly first mentioned in the introduction to this report.

The string assignment and modification problem is described, by example, in figure 4.2. As you can see, four variables were affected when the user modified on byte in the string "B". The real problem is when program logic is designed expecting the variables "A", "C", and "D" to contain the string as it appeared at the time of the assignment. It is possible to spend considerable time attempting to determine why the program is not functioning properly when the data is thought to be correct.

The anomaly has been corrected in the present version of the XPL compiler by forcing a string-move into a separate area of the free string space for each variable which receives a string descriptor, except for the OUTPUT pseudo-variable. Therfore, the assignment performed in the figure 4.2 example would cause three

Contents of the descriptors and associated string area before modification:

Variable	Descriptor	Contents
A	"00000000"	Null
B	"05010EA4"	ABCDEF
C	"00000000"	Null
D	"0A010EB0"	12345*ABCDE

XPL statements which modify the strings:

```
A,B,C = D;

BTYE(B,5) = ' ';
```

Contents of the descriptors and associated string area after the modification:

All Descriptors = "0A010EB0"

String Contents = 12345 ABCDE

Figure 4.2 Character-string Anomaly

identical copies of the string to be generated, one for each variable on the left of the assignment operator.

The string move problem was not the only reason for creating unique copies of strings. Assigning strings to and from common character-string areas also requires special string movement. This is done so all string descriptors will reference addresses within its associated free string space. The string compaction routine, XPLPACK, also requires that all descriptors address an area between the values contained in the string preamble variables FREEBASE and FREELIMIT.

Since string movement during compaction is done using the addresses contained in the descriptors, if the address was not within the areas stated, unreliable and probably disastrous results would occur. Therefore, to ensure proper function of this procedure, all assignments to or from a common character-string, along with character-strings referenced as parameters to or from external modules are copied to the receiving free string area and the descrptors are modified to reflect the address within this area.

Character-String Compaction

The enhancement in the current XPL system to allow any number of character-string areas, made it necessary to modify the XPL version III character-string compaction module "COMPACTIFY" to become a generalized routine capable of compressing any free string space.

The critical addresses needed by the version IV compaction routine, XPLPACK, are contained in the descriptor array for each string area defined, both common and internal. When it has been determined that string compaction is required (FREEPOINT >= FREELIMIT), The XPLPACK public procedure is called with register 0 containing the base address of the string area requiring compression.

The XPLPACK routine, when activated, will load the register 0 value into register 4 prior to the string compaction process. This is performed, because the variable 'DESCRIPTOR' will be referenced to obtain the required address, and is predefined in the XCOM symbol table with the SYBASE variable initialized to register 4.

During the process of string compaction, the variables LOWER_BOUND, FREEPOINT, FREEBASE, FREELIMIT, and NDESCRIPT will be referenced and are equated (using LITERALLYs) to refer to the values in the preamble of the array DESCRIPTOR (see appendix A4).

E. XPL Submonitor

Within the version III XPL system, the submonitor performed a major function. It was responsible for allocating and deallocating memory for the XPL program before and after program execution; loading the actual XPL program, data, and string areas into memory (the Linkage Editor was not used); provide the initial and final interface between the XPL program and the operating system, along with being the only interface between the XPL program and the IBM data management 'access methods' (BSAM and

QSAM). Also, the submonitor provided the linkage between the user program and the "TRACE" facility.

The submonitor was the outer shell or limiting boundary to the XPL program, even to the extent that an XPL program could only be executed by invoking the submonitor via the "PGM=XPLSM" parameter on the JCL EXEC statement and referencing the user's program as a data set for input to the submonitor ("PROGRAM").

The present submonitor has changed considerably, both in function and purpose.

The submonitor no longer performs the storage management and program loading roll. This is now provided by the Linkage Editor. The entire module has been modified to interface with the XPL program as an externally referenced module to provide services which cannot be performed by the user program with any degree of efficiency (Input/Output).

The interface with the execution trace facility has been removed, unfortunately. This was done because the TRACE routine requires that register 15 always contain the address of the submonitor's entry address, which can not be provided with the version IV implementation of external modules.

V. Conclusion

A. Recommendations for Further Work

The enhancements provided by this project extend the versatility of the XPL compiler generator system, but additional modification should be performed to remove the limitations identified in the introduction to this report. Specifically, providing the 'SUBSTR' pseudo-variable as the target of an assignment, and the creation of data 'structures' need to be included.

Furthermore, the following recommendations, concerning the present implementation of the XCOM compiler, require additional analysis.

One limitation, in the area of common data variables, is the restriction that any variable declared with the attribute 'COMMON', may not be initialized at compile-time. The reason is, that all linkage editor CSECTs are created at the end of the compilation, and the initialization of common variables would require a separate CSECT (as is done with common character-strings). Therefore, a problem arises as to the storage of initial values until the CSECTS are generated. Saving these values in memory is out of the question, and storing the values in a work file (FILE2 or FILE3) would present considerable difficulty during creation of the TXT record for the data and string CSECTs by attempting to 'sort out' which area within the files belong to which CSECT.

A reasonable solution would be to modify the XPL submonitor to accept another work file (FILE4), which would contain the initial values. Also, additional procedures (similar to the string file handling procedures 'EMITCHAR' and 'GETSTRINGS') would be needed within XCOM. If this technique were implemented, the relative lower and upper bounds of the initial values in the work file could be contained in the symbol table. Another alternative may be to create the initial value CSECTs during the compilation process. This solution would alleviate the 'FILE4' requirement. However, careful modification of the XCOM procedure 'SETINIT' and additional procedures to create the different types of common CSECTs would be required (i.e. BIT(8), BIT(16), FIXED, CHARACTER).

Another area which deserves attention, is within the procedure 'PUNCH_TXT', where the data is moved from the work array 'CODE' to the array 'BINARY'.

The present technique moves one byte for each iteration of a DO-loop. Within the loop are two IF statements which are tested at each iteration. This is rather inefficient.

A more efficient technique, would be to create a character-string descriptor addressing the array CODE, which contains the data to be moved. It would then be possible to use the 'SUBSTR' function to move the maximum number of bytes allowable (56 bytes) into the TXT record area in BINRCD, which is already defined as a character-string through declaration.

One should be cautioned, however, that the maximum size of a character-string is 256 bytes and the size of the array CODE is

presently 3600 bytes. The address within the descriptor pointing to the CODE array would require modification for each 256 bytes moved from this area.

Since the "TRACE" facility has been lost in the version IV implementation, a viable replacement should be offered. A suggestion is to incorporate, into the XPL submonitor, the capability to take a "SNAP" dump during the execution of the XPL program, along with including the required interface within the analysis procedures of XCOM. Then, even though a run-time execution trace is not provided, a dump of the working registers, selected area of the program, and the status of the external data sets may be obtained.

The last recommendation concerns the structure of the XPL compiler itself. The present version of XCOM should not be referred to as the version IV compiler (perhaps version III 1/2), because the compiler has not been recompiled using itself. Therefore, it is still a product of t he version III compiler and can not use any of the extended facilities which the version IV compiler produces.

The XCOM compiler should be recompiled but, before it is, a careful analysis should be performed to determine which procedures are to comprise the external modules in the new structure, and what data variables are to become common.

B. Advice to the Users of the XPL System

The XPL compiler generator system is a very reliable series of program modules. With the exception of one trivial "bug", the XCOM compiler used, during this project, to generate the version IV XCOM has produced totally error free object code. Hopefully, the version IV XCOM compiler will maintain this standard for correctness. The user of either system should be confident that the compiler will produce acceptable object code.

The only other advice to the users of the XPL system, is to remind them that XPL is not a programming language one is probably accustomed to. If one is very knowledgeable of PL/I, it should be remembered that XPL is not PL/I, and just because XPL does not "act" like PL/I does not mean there is a bug in the XPL compiler. If you take a little time to understand what the XPL language is attempting to accomplish, you will benefit in two ways: First, you will learn another implementation of a programming language and, the task of creating "your" compiler will be a more pleasurable and rewarding experience.

References

1. McKeeman, William M., Horning, James J., Wortman, David B., A Compiler Generator, 1970, Prentice-Hall Inc.

2. IBM Corporation, OS/VS Linkage Editor and Loader, GC26-3813-4, 1975.

3. Payne William H., Machine Assembly and Systems Programming for the IBM 360, 1969, Harper & Row.

4. Intel Corporation, PL/M-80 Programming Manual, 98-268B, 1977.

5. IBM Corporation, OS/ PL/1 Optimizing Compiler: Program Logic Manual, LY33-6008, 1973.

APPENDICES

A1	XPLXC	Compile JCL Procedure
A2	XPLXCL	Compile and Link JCL Procedure
A3	XPLXCLG	Compile, Link, and Go JCL Procedure
A4	XPLPACK	Source Program Listing
A5	XPLMON	Source Program Listing
A6	XCOM	Source Program Listing

```
//XPLXC     PROC    C=140
//XPL         EXEC    PGM=XPLSM,REGION=&C.K
//STEPLIB     DD      DSN=SYS3.LINKLIB3,DISP=SHR
//SYSPRINT    DD      SYSOUT=A
//PROGRAM     DD      DSN=SYS1.XCOM,UNIT=SYSDA,VOL=SER=PGMLIB,
//      DISP=SHR
//FILE1       DD      UNIT=SYSDA,SPACE=(TRK,(40,10),RLSE)
//FILE2       DD      UNIT=(SYSDA,SEP=(FILE1)),
//      SPACE=(TRK,(40,10),RLSE)
//FILE3       DD      UNIT=SYSDA,SPACE=(TRK,(40,10),RLSE)
//SYSPUNCH    DD      DSN=&&LOADSET,DISP=(MOD,PASS),
//      SPACE=(800,(200,100)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
//      UNIT=(SYSDA,SEP=(FILE1,FILE2))
//SYSIN       DD      DDNAME=SOURCE
// PEND
```

```
//XPLXCL      PROC    C=140,L=96
//XPL          EXEC     PGM=XPLSM,REGION=&C.K
//STEPLIB      DD       DSN=SYS3.LINKLIB3,DISP=SHR
//SYSPRINT     DD       SYSOUT=A
//PROGRAM      DD       DSN=SYS1.XCOM,UNIT=SYSDA,VOL=SER=PGMLIB,
//      DISP=SHR
//FILE1        DD       UNIT=SYSDA,SPACE=(TRK,(40,10),RLSE)
//FILE2        DD       UNIT=(SYSDA,SEP=(FILE1)),
//      SPACE=(TRK,(40,10),RLSE)
//FILE3        DD       UNIT=SYSDA,SPACE=(TRK,(40,10),RLSE)
//SYSPUNCH     DD       DSN=&&LOADSET,DISP=(MOD,PASS),
//      SPACE=(800,(200,100)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
//      UNIT=(SYSDA,SEP=(FILE1,FILE2))
//SYSIN        DD       DDNAME=SOURCE
//LKED       EXEC     PGM=IEWL,PARM='LIST,LET',REGION=&L.K,
//     COND=(0,NE,XPL)
//SYSPRINT     DD       SYSOUT=A
//SYSLIN       DD       DSN=&&LOADSET,DISP=(OLD,DELETE)
//               DD       DDNAME=OBJECT
//SYSLMOD      DD      DSN=&&GOSET(GO),UNIT=SYSDA,SPACE=(1024,(50,20,1)),
//     DISP=(MOD,PASS)
//SYSLIB       DD      DSN=SYS1.XPLLIB,DISP=SHR,UNIT=SYSDA,VOL=SER=PGMLIB
//SYSUT1       DD       DSN=&SYSUT1,UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),
//     SPACE=(1024,(50,20))
// PEND
```

```
//XPLXCLG      PROC    C=140,L=96,G=64
//XPL          EXEC    PGM=XPLSM,REGION=&C.K
//STEPLIB      DD      DSN=SYS3.LINKLIB3,DISP=SHR
//SYSPRINT     DD      SYSOUT=A
//PROGRAM      DD      DSN=SYS1.XCOM,UNIT=SYSDA,VOL=SER=PGMLIB,
//      DISP=SHR
//FILE1        DD      UNIT=SYSDA,SPACE=(TRK,(40,10),RLSE)
//FILE2        DD      UNIT=(SYSDA,SEP=(FILE1)),
//      SPACE=(TRK,(40,10),RLSE)
//FILE3        DD      UNIT=SYSDA,SPACE=(TRK,(40,10),RLSE)
//SYSPUNCH     DD      DSN=&&LOADSET,DISP=(MOD,PASS),
//      SPACE=(800,(200,100)),DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
//      UNIT=(SYSDA,SEP=(FILE1,FILE2))
//SYSIN        DD      DDNAME=SOURCE
//LKED       EXEC    PGM=IEWL,PARM='LIST,LET',REGION=&L.K,
//     COND=(0,NE,XPL)
//SYSPRINT     DD      SYSOUT=A
//SYSLIN       DD      DSN=&&LOADSET,DISP=(OLD,DELETE)
//               DD      DDNAME=OBJECT
//SYSLMOD      DD      DSN=&&GOSET(GO),UNIT=SYSDA,SPACE=(1024,(50,20,1)),
//     DISP=(MOD,PASS)
//SYSLIB       DD      DSN=SYS1.XPLLIB,DISP=SHR,UNIT=SYSDA,VOL=SER=PGMLIB
//SYSUT1       DD      DSN=&SYSUT1,UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),
//     SPACE=(1024,(50,20))
//GO           EXEC    PGM=*.LKED.SYSLMOD,COND=((0,NE,XPL),(4,LT,LKED)),
//     REGION=&G.K
//SYSPRINT     DD      SYSOUT=A
// PEND
```

```
PUBLIC: XPLPACK:
   PROCEDURE;
   DECLARE (I, J, K, L, ND, TC, BC, DELTA) FIXED;
   DECLARE DX_SIZE LITERALLY '500', DX(DX_SIZE) BIT(16);
   DECLARE MASK FIXED INITIAL ("FFFFFF"),  TRIED BIT(1);
  /* SET UP VARIABLE NAMES TO LINK WITH THE REPRESENTATIVE VALUES
     IN THE PARTICULAR STRING'S PROLOGUE WHICH IS BEING PACKED  */
   DECLARE LOWER_BOUND LITERALLY 'DESCRIPTOR(0)';
   DECLARE FREEPOINT LITERALLY 'DESCRIPTOR(1)';
   DECLARE FREEBASE  LITERALLY 'DESCRIPTOR(2)';
   DECLARE FREELIMIT LITERALLY 'DESCRIPTOR(3)';
   DECLARE NDESCRIPT LITERALLY 'DESCRIPTOR(4)';

  /* NOW SET THE BASE OF THE ARRAY DESCRIPTOR WITH THE STRING BASE
     VALUE FROM REG 0 INTO REG 4  ( SEE SYMBOL TABLE "DESCRIPTOR" ) */
   CALL INLINE("18",4,0);  /*LOAD R4 FROM R0 */

   /* SET THE LOWER BOUND OF THE COLLECTABLE AREA */
   IF LOWER_BOUND = 0 THEN LOWER_BOUND = FREEBASE;
 DO TRIED = 0 TO 1;
   ND = -1;
   /* FIND THE COLLECTABLE DESCRIPTORS  */
   DO I = 5 TO NDESCRIPT-1;
      IF (DESCRIPTOR(I) & MASK) >= LOWER_BOUND THEN
         DO;
            ND = ND + 1;
            IF ND > DX_SIZE THEN
               DO;  /* WE HAVE TOO MANY POTENTIALLY COLLECTABLE STRINGS  */
                  OUTPUT = '* * * NOTICE FROM XPLPACK:     DISASTROUS STRING OVE
RFLOW.    JOB ABANDONED. * * *';
                  CALL EXIT;
               END;
            DX(ND) = I;
         END;
   END;
   /* SORT IN ASCENDING ORDER  */
   K, L = ND;
   DO WHILE K <= L;
      L = -2;
      DO I = 1 TO K;
         L = I - 1;
         IF (DESCRIPTOR(DX(L)) & MASK) > (DESCRIPTOR (DX(I)) & MASK) THEN
            DO;
               J = DX(L); DX(L) = DX(I); DX(I) = J;
               K = L;
            END;
      END;
   END;
   /* MOVE THE ACTIVE STRINGS DOWN  */
   FREEPOINT = LOWER_BOUND;
   TC, DELTA = 0;
   BC = 1;   /* SETUP INITIAL CONDITION  */
   DO I = 0 TO ND;
      J = DESCRIPTOR(DX(I));
      IF (J & MASK) - 1 > TC THEN
```

```
            DO;
              IF DELTA > 0 THEN
               DO K = BC TO TC;
                  COREBYTE(K-DELTA) = COREBYTE(K);
               END;
               FREEPOINT = FREEPOINT + TC - BC + 1;
               BC = J & MASK;
               DELTA = BC - FREEPOINT;
            END;
         DESCRIPTOR (DX(I)) = J - DELTA;
         L = (J & MASK) + SHR(J, 24);
         IF TC < L THEN TC = L;
      END;
      DO K = BC TO TC;
         COREBYTE(K-DELTA) = COREBYTE(K);
      END;
      FREEPOINT = FREEPOINT + TC - BC + 1;
      IF SHL(FREELIMIT-FREEPOINT, 4) < FREELIMIT-FREEBASE THEN
         LOWER_BOUND = FREEBASE;
      ELSE
         DO;
            LOWER_BOUND = FREEPOINT;
            RETURN;
         END;
      /* THE HOPE IS THAT WE WON'T HAVE TO COLLECT ALL STRINGS EVERY TIME */
      END ;  /* OF THE DO TRIED LOOP        */
      IF FREELIMIT-FREEPOINT < 256 THEN
               DO;
                  OUTPUT = '* * * NOTICE FROM XPLPACK:     INSUFFICIENT STRING SPAC
E  JOB ABANDONED. * * *';
                  CALL EXIT;     /* FORCE ABEND  */
               END;

END XPLPACK;
  EOF
```

```
XMON     TITLE 'XPLSM SUBMONITOR FOR THE XPL COMPILER SYSTEM'
         SPACE 2
*********************************************************************
*                                                                   *
*                                                                   *
*                                                                   *
*        XPLMON   SUBMONITOR FOR THE XPL COMPILER GENERATOR SYSTEM   *
*
*                 MODIFIED FOR VERSION IV OF THE XPL COMPILER "XCOM"
*
*                                                                   *
*                                                                   *
*                                   DAVID B. WORTMAN                *
*                                   STANFORD UNIVERSITY             *
*                                   MARCH  1969                     *
*                                                                   *
*                                                                   *
*********************************************************************
         EJECT
         SPACE 1
*********************************************************************
*                                                                   *
*                                                                   *
*                                                                   *
*        DEFINE PARAMETRIC CONSTANTS FOR XPLSM                      *
*                                                                   *
*                                                                   *
*********************************************************************
         SPACE 1
         GBLA  &INPUTS              NUMBER OF INPUT FILES
         GBLA  &OUTPUTS             NUMBER OF OUTPUT FILES
         GBLA  &FILES               NUMBER OF DIRECT ACCESS FILES
         LCLA  &I                   VARIABLE FOR ITERATION LOOPS
         SPACE 5
IOPACK   CSECT
         SPACE 2
&INPUTS  SETA  3                    INPUT(I),    I = 0,1,2,3
         SPACE 1
&OUTPUTS SETA  3                    OUTPUT(I),   I = 0,1,2,3
         SPACE 1
&FILES   SETA  3                    FILE(I,*),   I = 1,2,3
         SPACE 1
*********************************************************************
*                                                                   *
*                                                                   *
*        FILEBYTS DETERMINES THE BLOCKSIZE FOR DIRECT ACCESS FILE   *
*        I/O.  IT SHOULD BE EQUAL TO THE VALUE OF THE LITERAL       *
*        CONSTANT 'DISKBYTES' IN THE XCOM COMPILER FOR COMPILATION  *
*        TO WORK SUCCESSFULLY.  THE VALUE OF FILEBYTS WHICH IS      *
*        ASSEMBLED IN MAY BE OVERIDDEN BY THE 'FILE=NNNN' PARAMETER *
*        ON THE OS/360 EXEC CARD.                                   *
*                                                                   *
*                                                                   *
*                                                                   *
*        DEVICE       ALLOWABLE RANGE         SUGGESTED VALUE      *
```

```
*                                                                     *
*       2311      80 <= FILEBYTS <= 3624        3600                *
*                                                                     *
*       2314      80 <= FILEBYTS <= 7294        7200                *
*                                                                     *
*       2321      80 <= FILEBYTS <= 2000        2000                *
*                                                                     *
*                                                                     *
*       LARGER VALUES MAY BE USED FOR FILEBYTS IF THE SUBMONITOR      *
*       IS REASSEMBLED WITH  RECFM=FT  SPECIFIED IN THE DCBS FOR      *
*       THE DIRECT ACCESS FILES.                                      *
*                                                                     *
*                                                                     *
***********************************************************************
        SPACE 2
FILEBYTS EQU   3600                2311  DISKS
        SPACE 2
***********************************************************************
*                                                                     *
*                                                                     *
*       BLKSIZE DEFAULT FOR SOME INPUT AND OUTPUT FILES.  SEE THE     *
*       EXIT LIST HANDLING ROUTINE GENXT.  SHOULD BE THE LARGEST      *
*       MULTIPLE OF 80 THAT IS LESS THAN OR EQUAL TO FILEBYTS         *
*                                                                     *
***********************************************************************
        SPACE 2
IOFBYTS  EQU   80*(FILEBYTS/80)
        SPACE 2
***********************************************************************
*                                                                     *
*                                                                     *
*       DEFINE THE REGISTERS USED TO PASS PARAMETERS TO THE           *
*       SUBMONITOR FROM THE XPL PROGRAM                               *
*                                                                     *
*                                                                     *
***********************************************************************
        SPACE 2
SVCODE   EQU   1                   CODE INDICATING SERVICE REQUESTED
        SPACE 1
PARM1    EQU   0                   FIRST PARAMETER
        SPACE 1
PARM2    EQU   2                   SECOND PARAMETER
        SPACE 2
***********************************************************************
*                                                                     *
*                                                                     *
*       DEFINE THE SERVICE CODES USED BY THE XPL PROGRAM TO           *
*       INDICATE SERVICE REQUESTS TO THE SUBMONITOR                   *
*                                                                     *
*                                                                     *
***********************************************************************
        SPACE 3
GETC     EQU   4                   SEQUENTIAL INPUT
        SPACE 1
PUTC     EQU   8                   SEQUENTIAL OUTPUT
        SPACE 1
```

```
TRC       EQU   12                  INITIATE TRACING
          SPACE 1
UNTR      EQU   16                  TERMINATE TRACING
          SPACE 1
EXDMP     EQU   20                  FORCE A CORE DUMP
          SPACE 1
GTIME     EQU   24                  RETURN TIME AND DATE
          SPACE 1
RSVD1     EQU   28                  (UNUSED)
          SPACE 1
RSVD2     EQU   32                  CLOCK_TRAP          (NOP IN XPLSM)
          SPACE 1
RSVD3     EQU   36                  INTERRUPT_TRAP      (NOP IN XPLSM)
          SPACE 1
RSVD4     EQU   40                  MONITOR             (NOP IN XPLSM)
          SPACE 1
RSVD5     EQU   44                  (UNUSED)
          SPACE 1
RSVD6     EQU   48                  (UNUSED)
          SPACE 1
          SPACE 2
*****************************************************************************
*                                                                           *
*         GENERATE THE SERVICE CODES FOR DIRECT ACCESS FILE I/O             *
*         BASED ON THE NUMBER OF FILES AVAILABLE  (&FILES)                  *
*                                                                           *
*****************************************************************************
          SPACE 2
FILEORG   EQU   RSVD6+4             ORIGIN FOR THE FILE SERVICE CODES
          SPACE 1
&I        SETA  1
.SC1      AIF   (&I GT &FILES).SC2
RD&I      EQU   FILEORG+8*(&I-1)    CODE FOR READING FILE&I
WRT&I     EQU   FILEORG+8*(&I-1)+4  CODE FOR WRITING FILE&I
&I        SETA  &I+1
          AGO   .SC1
.SC2      ANOP
          SPACE 2
ENDSERV   EQU   FILEORG+8*(&I-1)    1ST UNUSED SERVICE CODE
          SPACE 2
*****************************************************************************
*                                                                           *
*                                                                           *
*         DEFINE REGISTER USAGE                                             *
*                                                                           *
*****************************************************************************
          SPACE 2
RTN       EQU   3                   REGISTER CONTAINING COMPLETION
*                                   CODE RETURNED BY THE PROGRAM
          SPACE 1
EBR       EQU   10                  BASE REGISTER USED DURING
*                                   INITIALIZATION
          SPACE 1
CBR       EQU   14                  LINKAGE REGISTER USED FOR CALLS
*                                   TO THE SUBMONITOR
          SPACE 1
```

```
SELF     EQU   15                  REGISTER THAT ALWAYS CONTAINS
*                                  THE ADDRESS OF THE SUBMONITOR
*                                  ENTRY POINT
         SPACE 1
*********************************************************************
*                                                                   *
*        BIT CONSTANTS NEEDED FOR CONVERSING WITH OS/360 DCBS       *
*                                                                   *
*********************************************************************
         SPACE 2
OPENBIT  EQU   B'00010000'          DCBOFLGS BIT INDICATING OPEN
*                                  SUCCESSFULLY COMPLETED
         SPACE 1
TAPEBITS EQU   B'10000001'          BITS INDICATING A MAGNETIC TAPE
         SPACE 1
KEYLBIT  EQU   B'00000001'          BIT IN RECFM THAT INDICATES
*                                  KEYLEN WAS SET EXPLICITELY ZERO
         SPACE 2
*********************************************************************
*                                                                   *
*        FLAG BITS USED TO CONTROL SUBMONITOR OPERATION             *
*                                                                   *
*********************************************************************
         SPACE 2
ALLBITS  EQU   B'11111111'          MASK
         SPACE 1
TRACEBIT EQU   B'10000000'          BEGIN EXECUTION OF THE PROGRAM
*                                  IN TRACE MODE
         SPACE 1
SFILLBIT EQU   B'01000000'          1 CHARACTER OF FILL NEEDED BY
*                                  THE PUT ROUTINE
         SPACE 1
LFILLBIT EQU   B'00100000'          LONGER FILL NEEDED BY PUT ROUTINE
         SPACE 1
DUMPBIT  EQU   B'00001000'          GIVE A CORE DUMP FOR I/O ERRORS
         SPACE 2
*********************************************************************
*                                                                   *
*                                                                   *
*              DEFINE ABEND CODES ISSUED BY THE SUBMONITOR          *
*                                                                   *
*********************************************************************
         SPACE 2
OPENABE  EQU   100                  UNABLE TO OPEN ONE OF THE FILES:
*                                  PROGRAM, SYSIN, OR SYSPRINT
         SPACE 1
PGMEOD   EQU   200                  UNEXPECTED END OF FILE WHILE
*                                  READING IN THE XPL PROGRAM
         SPACE 1
PGMERR   EQU   300                  SYNAD ERROR WHILE READING IN
*                                  THE XPL PROGRAM
         SPACE 1
COREABE  EQU   400                  XPL PROGRAM WON'T FIT IN
*                                  THE AMOUNT OF MEMORY AVAILABLE
         SPACE 1
CODEABE  EQU   500                  INVALID SERVICE CODE FROM THE
```

```
*                                       XPL PROGRAM
          SPACE 1
OUTSYND   EQU   800                     SYNAD ERROR ON OUTPUT FILE
          SPACE 1
PFABE     EQU   900                     INVALID OUTPUT FILE SPECIFIED
          SPACE 1
INSYND    EQU   1000                    SYNAD ERROR ON INPUT FILE
          SPACE 1
INEODAB   EQU   1200                    END OF FILE ERROR ON INPUT FILE
          SPACE 1
GFABE     EQU   1400                    INVALID INPUT FILE SPECIFIED
          SPACE 1
FLSYND    EQU   2000                    SYNAD ERROR ON DIRECT ACCESS FILE
          SPACE 1
FLEOD     EQU   2200                    END OF FILE ERROR ON DIRECT
*                                       ACCESS FILE
          SPACE 1
USERABE   EQU   4000                    XPL PROGRAM CALLED EXIT TO
*                                       FORCE A CORE DUMP
          EJECT
          SPACE 5
*********************************************************************
*                                                                   *
*                                                                   *
*                                                                   *
*         SUBMONITOR  INITIALIZATION                                *
*                                                                   *
*                                                                   *
*********************************************************************
          SPACE 3
          ENTRY XPLMON                  WHERE OS ENTERS THE SUBMONITOR
          SPACE 2
          DS    0F
          USING *,15
          SPACE 1
XPLMON    SAVE  (14,12),T,*             SAVE ALL REGISTERS
          ST    13,SAVE+4               SAVE RETURN POINTER
          LA    15,SAVE                 ADDRESS OF SUBMONITOR'S OS SAVE AREA
          ST    15,8(0,13)
          LR    13,15
          USING SAVE,13
          MVI   FLAGS,B'00000000'       RESET ALL FLAGS
          B     ENTRY                   PROCESS REQUEST
*********************************************************************
*                                                                   *
*                                                                   *
*                                                                   *
*         CONSTANTS USED DURING INITIALIZATION                      *
*                                                                   *
*                                                                   *
*********************************************************************
          SPACE 2
SAVE      DS    18F                     SAVE AREA FOR OS/360
ACORE     DS    A                       ADDRESS OF CORE FOR THE PROGRAM
CORESIZE  DS    F                       SIZE OF CORE IN BYTES
CONTROL   DS    A                       ADDRESS OF PARAMETER STRING PASSED
```

```
*                                       TO THE SUBMONITOR BY OS/360
COREREQ  DS    0H                       CORE REQUEST CONTROL BLOCK
COREMIN  DC    F'110000'                MINIMUM AMOUNT OF CORE REQUIRED
COREMAX  DC    F'5000000'               MAXIMUM AMOUNT OF CORE REQUESTED
FREEUP   DC    A(2*FILEBYTS)            AMOUNT OF CORE TO RETURN TO OS
ALTFREE  DC    A(4*FILEBYTS)            AMOUNT OF CORE FREED FOR ALTER
         SPACE 1
*****************************************************************************
*                                                                           *
*        BLOCK OF PARAMETERS PASSED TO THE XPL PROGRAM                      *
*                                                                           *
*****************************************************************************
         SPACE 1
PGMPARMS DS    F                        R0  UNUSED
CORETOP  DC    A(0)                     R1  ADDRESS OF TOP OF CORE
CODEADR  DC    F'0'                     R2  ADDRESS OF START OF 1ST RECORD
*                                       OF THE XPL PROGRAM
DATADR   DC    F'0'                     R3  ADDRESS OF THE START OF THE XPL
*                                           PROGRAM'S DATA AREA
         SPACE 1
TRCM     DC    CL5'TRACE'
ALTRM    DC    CL5'ALTER'
CMNM     DC    CL4'MIN='
CMXM     DC    CL4'MAX='
FREEM    DC    CL5'FREE='
DMPM     DC    CL4'DUMP'
FILEM    DC    CL5'FILE='
         EJECT
         SPACE 5
*****************************************************************************
*                                                                           *
*                                                                           *
*        ROUTINES TO PROVIDE DEFAULT DATASET INFORMATION IF NONE            *
*        IS PROVIDED BY JCL OR VOLUME LABELS.  IN PARTICULAR,               *
*        BLKSIZE, LRECL, BUFNO, AND RECFM INFORMATION.                      *
*                                                                           *
*                                                                           *
*        EXIT LISTS FOR DCBS                                                *
*                                                                           *
*****************************************************************************
         SPACE 2
         DS    0F
INEXIT0  DC    X'85'                    INPUT0
         DC    AL3(INXT0)
         SPACE 1
OUTEXIT0 DC    X'85'                    OUTPUT0
         DC    AL3(OUTXT0)
         SPACE 1
INEXIT2  DC    X'85'                    INPUT2
         DC    AL3(INXT2)
         SPACE 1
OUTEXIT2 EQU   INEXIT0                  OUTPUT2
         SPACE 2
&I       SETA  3
.IDF1    AIF   (&I GT &INPUTS).IDF2
INEXIT&I EQU   INEXIT2                  INPUT&I
```

```
&I       SETA  &I+1
         AGO   .IDF1
.IDF2    ANOP
         SPACE 1
&I       SETA  3
.ODF1    AIF   (&I GT &OUTPUTS).ODF2
OUTEXIT&I EQU  INEXIT2
&I       SETA  &I+1
         AGO   .ODF1
.ODF2    ANOP
         SPACE 1
*********************************************************************
*                                                                   *
*        DCB EXIT ROUTINE ENTRY POINTS                              *
*                                                                   *
*********************************************************************
         SPACE 2
INXT0    MVC   DEFAULTS(6),INDFLT0
         B     GENXT
         SPACE 1
INXT2    MVC   DEFAULTS(6),INDFLT2
         B     GENXT
         SPACE 1
OUTXT0   MVC   DEFAULTS(6),OUTDFLT0
         SPACE 1
*********************************************************************
*                                                                   *
*                                                                   *
*        DCB EXIT LIST PROCESSING ROUTINE FOR OPEN EXITS            *
*                                                                   *
*********************************************************************
         SPACE 2
GENXT    DS    0H
         USING IHADCB,1              REGISTER 1 POINTS AT THE DCB
         NC    DCBBLKSI,DCBBLKSI     CHECK BLKSIZE
         BNZ   GXT1                  ALREADY SET
         MVC   DCBBLKSI(2),DFLTBLKS
*                                    PROVIDE DEFAULT BLOCKSIZE
         SPACE 1
GXT1     NC    DCBLRECL,DCBLRECL     CHECK LRECL
         BNZ   GXT2                  ALREADY SET
         MVC   DCBLRECL(2),DFLTLREC
*                                    PROVIDE DEFAULT LRECL
         SPACE 1
GXT2     CLI   DCBBUFNO,0            CHECK BUFNO
         BNE   GXT3                  ALREADY SPECIFIED
         MVC   DCBBUFNO(1),DFLTBUFN
*                                    PROVIDE DEFAULT BUFNO
         SPACE 1
GXT3     TM    DCBRECFM,ALLBITS-KEYLBIT
*                                    CHECK RECFM
         BCR   B'0111',14            ALREADY SET SO RETURN
         OC    DCBRECFM(1),DFLTRECF
*                                    PROVIDE DEFAULT RECFM
         BR    14                    RETURN
         DROP  1
```

```
         SPACE 2
*****************************************************************************
*                                                                           *
*        ARRAY OF DEFAULT ATTRIBUTES USED BY GENXT                          *
*                                                                           *
*****************************************************************************
         SPACE 1
DEFAULTS DS    0H
DFLTBLKS DS    1H                  DEFAULT BLKSIZE
DFLTLREC DS    1H                  DEFAULT LRECL
DFLTBUFN DS    AL1                 DEFAULT BUFNO
DFLTRECF DS    1BL1                DEFAULT RECFM
         SPACE 1
*****************************************************************************
*                                                                           *
*        DEFINE ATTRIBUTES PROVIDED FOR THE VARIOUS FILES                   *
*                                                                           *
*        INPUT(0), INPUT(1), OUTPUT(2)                                      *
*                                                                           *
*****************************************************************************
         SPACE 1
INDFLT0  DS    0H
         DC    H'80'               BLKSIZE=80
         DC    H'80'               LRECL=80
         DC    AL1(2)              BUFNO=2
         DC    B'10000000'         RECFM=F
         SPACE 1
*****************************************************************************
*                                                                           *
*        OUTPUT(0), OUTPUT(1)                                               *
*                                                                           *
*****************************************************************************
         SPACE 1
OUTDFLT0 DS    0H
         DC    H'133'              BLKSIZE=133
         DC    H'133'              LRECL=133
         DC    AL1(2)              BUFNO=2
         DC    B'10000100'         RECFM=FA
         SPACE 1
*****************************************************************************
*                                                                           *
*        INPUT(2), INPUT(3), OUTPUT(3)                                      *
*                                                                           *
*****************************************************************************
         SPACE 1
INDFLT2  DS    0H
         DC    AL2(IOFBYTS)        BLKSIZE=IOFBYTS
         DC    H'80'               LRECL=80
         DC    AL1(1)              BUFNO=1
         DC    B'10010000'         RECFM=FB
         EJECT
         SPACE 5
*****************************************************************************
*                                                                           *
*                                                                           *
*        INPUT - OUTPUT  ERROR ROUTINES                                     *
```

```
*                                                                     *
*                                                                     *
*                                                                     *
*       SYNAD AND EOD ERROR ROUTINES FOR INITIAL LOADING OF THE       *
*       XPL PROGRAM                                                   *
*                                                                     *
*                                                                     *
***********************************************************************
        SPACE 2
EODPGM  STM   0,2,ABEREGS          SAVE REGISTERS
        LA    1,PGMEOD             UNEXPECTED EOD WHILE READING IN
*                                  THE XPL PROGRAM
        B     ABEND                GO TO ABEND ROUTINE
        SPACE 2
ERRPGM  STM   0,2,ABEREGS          SAVE REGISTERS
        LA    1,PGMERR             SYNAD ERROR WHILE READING IN THE
*                                  XPL PROGRAM
        B     ABEND                GO TO ABEND ROUTINE
        SPACE 2
***********************************************************************
*                                                                     *
*                                                                     *
*       SYNAD AND EOD ROUTINES FOR INPUT(I),  I = 0,1, ...  ,&INPUTS  *
*                                                                     *
***********************************************************************
        SPACE 2
INEOD   L     2,SAVREG+PARM2*4     PICK UP SUBCODE SPECIFYING WHICH
*                                  INPUT FILE
        SLL   2,2                  SUBCODE*4
        L     2,GETDCBS(2)         FETCH DCB ADDRESS
        USING IHADCB,2
        ST    2,OCDCB              STORE IT FOR THE CLOSE SVC
        MVI   OCDCB,X'80'          FLAG END OF PARAMETER LIST
        CLOSE ,MF=(E,OCDCB)        CLOSE THE OFFENDING FILE
        SPACE 1
PCLOSE  DS    0H
        XC    DCBDDNAM,DCBDDNAM    MARK THE FILE PERMANENTLY UNUSABLE
        DROP  2
        B     RETNEOF              GO RETURN AN END OF FILE INDICATION
        SPACE 2
INSYNAD STM   0,2,ABEREGS          SAVE REGISTERS
        LA    1,INSYND             SYNAD ERROR ON AN INPUT FILE
        B     INERR                BRANCH TO ERROR ROUTINE
        SPACE 1
INEOD2  LA    1,INEODAB            EOD ON AN INPUT FILE
*                                  ATTEMPT TO READ AFTER AN EOD SIGNAL
INERR   A     1,SAVREG+PARM2*4     SUBCODE INDICATING WHICH INPUT FILE
        B     ABEND                BRANCH TO ABEND ROUTINE
        SPACE 2
***********************************************************************
*                                                                     *
*                                                                     *
*       SYNAD ERROR ROUTINES FOR OUTPUT FILES                         *
*                                                                     *
*                                                                     *
***********************************************************************
```

```
          SPACE 2
OUTSYNAD  STM   0,2,ABEREGS          SAVE REGISTERS
          LA    1,OUTSYND           SYNAD ERROR ON OUTPUT FILE
          B     INERR
          SPACE 2
*********************************************************************
*                                                                   *
*         SYNAD AND EOD ROUTINES FOR DIRECT ACCESS FILE I/O         *
*                                                                   *
*********************************************************************
          SPACE 2
FILESYND  STM   0,2,ABEREGS          SAVE REGISTERS
          LA    1,FLSYND            SYNAD ERROR ON DIRECT ACCESS FILE
          B     FILERR              GO TO ERROR ROUTINE
          SPACE 1
FILEEOD   STM   0,2,ABEREGS          SAVE REGISTERS
          LA    1,FLEOD             EOD ERROR ON DIRECT ACCESS FILE
          SPACE 1
FILERR    L     2,SAVREG+SVCODE*4   SERVICE CODE
          LA    0,RD1-8             COMPUTE WHICH DIRECT ACCESS FILE
          SR    2,0                 SERVICE_CODE - 1ST SERVICE CODE
          SRL   2,3                 DIVIDE BY 8
          AR    1,2
*                                   FALL THROUGH TO ABEND ROUTINE
          SPACE 2
*********************************************************************
*                                                                   *
*                                                                   *
*         ABEND ROUTINE FOR ALL I/O ERRORS                          *
*                                                                   *
*                                                                   *
*********************************************************************
          SPACE 2
ABEND     DS    0H
          ST    1,ABESAVE           SAVE ABEND CODE
          SPACE 1
          CLOSE (INPUT0,,OUTPUT0)   THESE MUST BE CLOSED FIRST
          CLOSE ,MF=(E,GETDCBS)     ATTEMPT TO CLOSE ALL FILES
          SPACE 1
          L     1,ABESAVE
          TM    FLAGS,DUMPBIT       IS A CORE DUMP DESIRED ?
          BZ    NODUMP              NO, ABEND QUIETLY
          SPACE 1
          ABEND (1),DUMP            ABEND WITH A DUMP
          SPACE 1
NODUMP    DS    0H
          ABEND (1)                 ABEND WITHOUT A DUMP
          SPACE 2
*********************************************************************
*                                                                   *
*                                                                   *
*         ROUTINE TO FORCE AN ABEND DUMP WHEN REQUESTED BY THE      *
*         XPL PROGRAM BY MEANS OF THE STATEMENT:                    *
*                                                                   *
*         CALL  EXIT  ;                                             *
*                                                                   *
```

```
*                                                                      *
************************************************************************
          SPACE 2
USEREXIT  DS    0H
          STM   0,2,ABEREGS          SAVE REGISTERS
          OI    FLAGS,DUMPBIT        FORCE A DUMP
          LA    1,USERABE            USER ABEND CODE
          B     ABEND                BRANCH TO ABEND
          EJECT
          SPACE 5
************************************************************************
*                                                                      *
*                                                                      *
*         DISPATCHER FOR ALL SERVICE REQUESTS FROM THE XPL PROGRAM     *
*                                                                      *
*                                                                      *
************************************************************************
          SPACE 2
ENTRY     DS    0H                   XPL PROGRAMS ENTER HERE
          STM   0,3,SAVREG           SAVE REGISTERS USED BY XPLSM
          SPACE 1
          LTR   SVCODE,SVCODE        CHECK THE SERVICE CODE FOR VALIDITY
          BNP   BADCODE              SERVICE CODE MUST BE > 0
          C     SVCODE,MAXCODE       AND < ENDSERV
          BH    BADCODE              GO ABEND
          SPACE 1
TABLE     B     TABLE(SVCODE)        GO DO THE SERVICE
          SPACE 1
          ORG   TABLE+GETC
          B     GET                  READ INPUT FILE
          SPACE 1
          ORG   TABLE+PUTC
          B     PUT                  WRITE OUTPUT FILE
          SPACE 1
          ORG   TABLE+TRC
          B     EXIT                 INITIATE TRACING OF THE PROGRAM
          SPACE 1
          ORG   TABLE+UNTR
          B     EXIT                 TERMINATE TRACING
          SPACE 1
          ORG   TABLE+EXDMP
          B     USEREXIT             TERMINATE WITH A CORE DUMP
          SPACE 1
          ORG   TABLE+GTIME
          B     GETIME               RETURN TIME AND DATE
          SPACE 1
          ORG   TABLE+RSVD1
          B     EXIT                 (UNUSED)
          SPACE 1
          ORG   TABLE+RSVD2
          B     EXIT                 CLOCK_TRAP        (NOP)
          SPACE 1
          ORG   TABLE+RSVD3
          B     EXIT                 INTERRUPT_TRAP    (NOP)
          SPACE 1
          ORG   TABLE+RSVD4
```

```
         B      EXIT                  MONITOR          (NOP)
         SPACE  1
         ORG    TABLE+RSVD5
         B      EXIT                  (UNUSED)
         SPACE  1
         ORG    TABLE+RSVD6
         B      EXIT                  (UNUSED)
         SPACE  2
*********************************************************************
*                                                                   *
*                                                                   *
*        DYNAMICALLY GENERATE THE DISPATCHING TABLE ENTRIES FOR     *
*        FILE I/O SERVICES.                                         *
*                                                                   *
*                                                                   *
*********************************************************************
         SPACE  2
&I       SETA   1                     LOOP INDEX
.DBR1    AIF    (&I GT &FILES).DBR2
*                                     FINISHED ?
         ORG    TABLE+RD&I
         B      READ                  BRANCH TO FILE READ ROUTINE
         ORG    TABLE+WRT&I
         B      WRITE                 BRANCH TO FILE WRITE ROUTINE
&I       SETA   &I+1                  INCREMENT COUNTER
         AGO    .DBR1                 LOOP BACK
.DBR2    ANOP
         SPACE  2
         ORG    TABLE+ENDSERV         RESET PROGRAM COUNTER
         SPACE  2
*********************************************************************
*                                                                   *
*                                                                   *
*        COMMON EXIT ROUTINE FOR RETURN TO THE XPL PROGRAM          *
*                                                                   *
*                                                                   *
*********************************************************************
         SPACE  2
EXIT     DS     0H
         LM     0,3,SAVREG            RESTORE REGISTERS
         L      13,SAVE+4
         STM    0,3,20(13)           STORE INTO USER AREA
         RETURN (14,12)               RETURN TO CALLER
         SPACE  2
*********************************************************************
*                                                                   *
*        ROUTINE TO ABEND IN CASE OF BAD SERVICE CODES              *
*                                                                   *
*********************************************************************
         SPACE  2
BADCODE  STM    0,2,ABEREGS           SAVE REGISTERS
         LA     1,CODEABE             BAD SERVICE CODE ABEND
         B      ABEND                 GO ABEND
         EJECT
         SPACE  5
*********************************************************************
```

```
*                                                                     *
*                                                                     *
*       INPUT ROUTINE FOR READING SEQUENTIAL INPUT FILES              *
*                                                                     *
*                                                                     *
*       INPUT TO THIS ROUTINE IS:                                     *
*                                                                     *
*     PARM1   ADDRESS OF THE NEXT AVAILABLE SPACE IN THE PROGRAMS     *
*             DYNAMIC STRING AREA  (FREEPOINT)                        *
*                                                                     *
*     SVCODE  THE SERVICE CODE FOR INPUT                              *
*                                                                     *
*     PARM2   A SUBCODE DENOTING WHICH INPUT FILE,                    *
*             INPUT(I),     I = 0,1, ... ,&INPUTS                     *
*                                                                     *
*       THE ROUTINE RETURNS:                                          *
*                                                                     *
*     PARM1   A STANDARD XPL STRING DESCRIPTOR POINTING AT THE INPUT  *
*             RECORD WHICH IS NOW AT THE TOP OF THE STRING AREA        *
*                                                                     *
*     SVCODE  THE NEW VALUE FOR FREEPOINT, UPDATED BY THE LENGTH OF   *
*             THE RECORD JUST READ IN                                 *
*                                                                     *
*                                                                     *
*       A STANDARD XPL STRING DESCRIPTOR HAS:                         *
*                                                                     *
*       BITS  0-7                 (LENGTH - 1) OF THE STRING        *
*       BITS  8-31                ABSOLUTE ADDRESS OF THE STRING    *
*                                                                     *
*                                                                     *
*                                                                     *
***********************************************************************
         SPACE 2
GET      DS    0H
         LA    SVCODE,&INPUTS      CHECK THAT THE SUBCODE IS VALID
         LTR   PARM2,PARM2         SUBCODE MUST BE >= 0
         BM    BADGET
         CR    PARM2,SVCODE        AND <= &INPUTS
         BH    BADGET              ILLEGAL SUBCODE
         SLL   PARM2,2             SUBCODE*4
         L     3,GETDCBS(PARM2)    ADDRESS OF DCB FOR THE FILE
         USING IHADCB,3
         NC    DCBDDNAM,DCBDDNAM   HAS THE FILE BEEN PERMANENTLY
*                                  CLOSED ?
         BZ    INEOD2              YES, SO TERMINATE THE JOB
         SPACE 1
         TM    DCBOFLGS,OPENBIT    IS THE FILE OPEN ?
         BO    GETOPEN             YES
         ST    3,OCDCB             STORE DCB ADDRESS FOR OPEN SVC
         MVI   OCDCB,X'80'         FLAG END OF PARAMETER LIST
         OPEN  ,MF=(E,OCDCB)       OPEN THE FILE
         LR    2,3                 COPY DCB ADDRESS
         TM    DCBOFLGS,OPENBIT    WAS THE FILE OPENED SUCCESSFULLY ?
         BZ    PCLOSE              NO, MARK FILE PERMANENTLY CLOSED AND
*                                  RETURN EOD INDICATION TO THE PROGRAM
         SPACE 2
```

```
GETOPEN  DS    0H
         GET   (3)                    LOCATE MODE GET
         SPACE 1
*****************************************************************************
*                                                                           *
*        USING LOCATE MODE, THE ADDRESS OF THE NEXT INPUT BUFFER            *
*        IS RETURNED IN R1                                                  *
*                                                                           *
*****************************************************************************
         SPACE 1
         L     2,SAVREG+PARM1*4       FETCH THE STRING DESCRIPTOR
         LA    2,0(,2)                ADDRESS PART ONLY
         LH    3,DCBLRECL             RECORD LENGTH
         DROP  3
         S     3,F1                   LENGTH - 1
         EX    3,GETMOVE              MOVE THE CHARACTERS
         ST    2,SAVREG+PARM1*4       BUILD UP A STRING DESCRIPTOR
         STC   3,SAVREG+PARM1*4       LENGTH FIELD
         LA    2,1(2,3)               NEW FREE POINTER
         ST    2,SAVREG+SVCODE*4
         B     EXIT                   RETURN TO THE XPL PROGRAM
         SPACE 2
*****************************************************************************
*                                                                           *
*        RETURN A NULL STRING DESCRIPTOR AS AN END OF FILE                  *
*        INDICATION THE FIRST TIME AN INPUT REQUEST FIND THE                *
*        END OF DATA CONDITION                                              *
*                                                                           *
*****************************************************************************
         SPACE 2
RETNEOF  DS    0H
         MVC   SAVREG+SVCODE*4(4),SAVREG+PARM1*4
*                                    RETURN FREEPOINT UNTOUCHED
         XC    SAVREG+PARM1*4(4),SAVREG+PARM1*4
*                                    RETURN A NULL STRING DESCRIPTOR
         B     EXIT                   RETURN TO THE XPL PROGRAM
         SPACE 2
*****************************************************************************
*                                                                           *
*        ROUTINE TO ABEND IN CASE OF AN INVALID SUBCODE                     *
*                                                                           *
*****************************************************************************
         SPACE 2
BADGET   STM   0,2,ABEREGS            SAVE REGISTERS
         LA    1,GFABE                INVALID GET SUBCODE
         B     INERR                  GO ABEND
         EJECT
         SPACE 5
*****************************************************************************
*                                                                           *
*                                                                           *
*                                                                           *
*        ROUTINE FOR WRITING SEQUENTIAL OUTPUT FILES                        *
*                                                                           *
*                                                                           *
*        INPUT TO THIS ROUTINE:                                             *
```

```
*                                                                     *
*       PARM1   XPL STRING DESCRIPTOR OF THE STRING TO BE OUTPUT      *
*                                                                     *
*       PARM2   SUBCODE INDICATING  OUTPUT(I),  I = 0,1, ... ,&OUTPUTS *
*                                                                     *
*       SVCODE  THE SERVICE CODE FOR OUTPUT                           *
*                                                                     *
*                                                                     *
*         THE STRING NAMED BY THE DESCRIPTOR IS PLACED IN THE NEXT    *
*         OUTPUT BUFFER OF THE SELECTED FILE.  IF THE STRING IS       *
*         SHORTER THAN THE RECORD LENGTH OF THE FILE THEN THE         *
*         REMAINDER OF THE RECORD IS PADDED WITH BLANKS.  IF THE      *
*         STRING IS LONGER THAN THE RECORD LENGTH OF THE FILE         *
*         THEN IT IS TRUNCATED ON THE RIGHT TO FIT.  IF THE SUBCODE   *
*         SPECIFIES OUTPUT(0) THEN A SINGLE BLANK IS CONCATENATED     *
*         ON TO THE FRONT OF THE STRING TO SERVE AS CARRIAGE CONTROL. *
*                                                                     *
*                                                                     *
***********************************************************************
          SPACE 2
PUT       DS    0H
          LTR   PARM2,PARM2          CHECK SUBCODE FOR VALIDITY
          BM    BADPUT               SUBCODE MUST BE >= 0
          LA    SVCODE,&OUTPUTS
          CR    PARM2,SVCODE         AND <= &OUTPUTS
          BH    BADPUT
          ST    PARM1,MOVEADR        SAVE THE STRING DESCRIPTOR
          SLL   PARM2,2              SUBCODE*4
          L     3,PUTDCBS(PARM2)     GET THE DCB ADDRESS
          USING IHADCB,3
          TM    DCBOFLGS,OPENBIT     IS THE FILE OPEN ?
          BO    PUTOPEN              YES, GO DO THE OUTPUT
          ST    3,OCDCB              STORE DCB ADDRESS FOR THE OPEN SVC
          MVI   OCDCB,X'8F'          FLAG END OF PARAMETER LIST AND SET
*                                    FLAG INDICATING OPENING FOR OUTPUT
          SPACE 1
          OPEN  ,MF=(E,OCDCB)        OPEN THE FILE
          SPACE 1
          TM    DCBOFLGS,OPENBIT     WAS THE OPEN SUCCESSFULL ?
          BZ    OUTSYNAD             NO, OUTPUT SYNAD ERROR
          SPACE 1
PUTOPEN   DS    0H
          PUT   (3)                  LOCATE MODE PUT
          SPACE 1
***********************************************************************
*                                                                     *
*         USING LOCATE MODE, THE ADDRESS OF THE NEXT OUTPUT BUFFER    *
*         IS RETURNED IN  R1.                                         *
*                                                                     *
***********************************************************************
          SPACE 1
          SR    15,15                CLEAR REGISTER 15
          C     15,MOVEADR           IS THE STRING NULL (DESCRIPTOR = 0)
          BE    NULLPUT              YES, SO PUT OUT A BLANK RECORD
          IC    15,MOVEADR           LENGTH-1 OF THE STRING
          LA    14,1(15)             REAL LENGTH OF THE STRING
```

```
         LH    0,DCBLRECL          RECORD LENGTH OF THE FILE
         LTR   PARM2,PARM2         CHECK SUBCODE FOR OUTPUT(0)
         BNZ   PUT1                NOT OUTPUT(0)
         LA    14,1(,14)           INCREASE REAL LENGTH BY ONE FOR
*                                  CARRIAGE CONTROL
PUT1     SR    0,14                RECORD LENGTH - REAL LENGTH
         BM    TOOLONG             RECORD LENGTH < REAL LENGTH
         BZ    MATCH               RECORD LENGTH = REAL LENGTH
*                                  RECORD LENGTH > REAL LENGTH
         OI    FLAGS,SFILLBIT+LFILLBIT
*                                  INDICATE PADDING REQUIRED
         S     0,F1                RECORD LENGTH - REAL LENGTH - 1
         BP    LONGMOVE            RECORD LENGTH - REAL LENGTH > 1
         NI    FLAGS,ALLBITS-LFILLBIT
*                                  RECORD LENGTH - REAL LENGTH = 1
*                                  IS A SPECIAL CASE
LONGMOVE ST    0,FILLENG           SAVE LENGTH FOR PADDING OPERATION
         B     MOVEIT              GO MOVE THE STRING
         SPACE 1
TOOLONG  LH    15,DCBLRECL         REPLACE THE STRING LENGTH
*                                  WITH THE RECORD LENGTH
         S     15,F1               RECORD LENGTH - 1 FOR THE MOVE
MATCH    NI    FLAGS,ALLBITS-SFILLBIT-LFILLBIT
*                                  INDICATE NO PADDING REQUIRED
         SPACE 1
MOVEIT   LTR   PARM2,PARM2         CHECK FOR OUTPUT(0)
         BZ    PUTZERO             OUTPUT(0) IS A SPECIAL CASE
         L     2,MOVEADR           STRING DESCRIPTOR
         LA    2,0(,2)             ADDRESS PART ONLY
         EX    15,MVCSTRNG         EXECUTE A MVC INSTRUCTION
         TM    FLAGS,SFILLBIT      IS PADDING REQUIRED ?
         BZ    EXIT                NO, RETURN TO THE XPL PROGRAM
         SPACE 1
         AR    1,15                ADDRESS TO START PADDING - 1
         MVI   1(1),C' '           START THE PAD
         TM    FLAGS,LFILLBIT      IS MORE PADDING REQUIRED ?
         BZ    EXIT                NO, RETURN TO XPL PROGRAM
         L     15,FILLENG          LENGTH OF PADDING NEEDED
         S     15,F1               LESS ONE FOR THE MOVE
         EX    15,MVCBLANK         EXECUTE MVC TO FILL IN BLANKS
         B     EXIT                RETURN TO THE XPL PROGRAM
         SPACE 1
*****************************************************************
*                                                               *
*        FOR A NULL STRING OUTPUT A BLANK RECORD                *
*                                                               *
*****************************************************************
         SPACE 1
NULLPUT  LH    15,DCBLRECL         RECORD LENGTH
         S     15,F2               LESS TWO FOR THE MOVES
         MVI   0(1),C' '           INITIAL BLANK
         EX    15,MVCNULL          EXECUTE MVC TO FILL IN THE BLANKS
         B     EXIT                RETURN TO THE XPL PROGRAM
         SPACE 1
*****************************************************************
*                                                               *
```

```
*        SPECIAL HANDLING FOR OUTPUT(0)                                 *
*                                                                       *
*************************************************************************
         SPACE 1
PUTZERO  L     2,MOVEADR          STRING DESCRIPTOR
         LA    2,0(,2)            ADDRESS PART ONLY
         EX    15,MVCP0           EXECUTE MVC TO MOVE THE STRING IN
         MVI   0(1),C' '          PROVIDE BLANK FOR CARRIAGE CONTROL
         TM    FLAGS,SFILLBIT     IS PADDING REQUIRED ?
         BZ    EXIT               NO, RETURN TO XPL PROGRAM
         AR    1,15               ADDRESS TO START PADDING - 1
         MVI   2(1),C' '          ONE BLANK TO START THE PAD
         TM    FLAGS,LFILLBIT     MORE PADDING REQUIRED ?
         BZ    EXIT               NO, RETURN TO THE XPL PROGRAM
         L     15,FILLENG         LENGTH OF PAD REQUIRED
         S     15,F1              LESS ONE FOR THE MOVE
         EX    15,MVCPOBL         EXECUTE MVC TO FILL IN THE BLANKS
         B     EXIT               RETURN TO THE XPL PROGRAM
         DROP  3
         SPACE 1
*************************************************************************
*                                                                       *
*        ROUTINE TO ABEND IN CASE OF AN INVALID SERVICE CODE            *
*                                                                       *
*************************************************************************
         SPACE 1
BADPUT   STM   0,2,ABEREGS        SAVE REGISTERS
         LA    1,PFABE            INVALID PUT SUBCODE
         B     INERR              GO ABEND
         EJECT
         SPACE 5
*************************************************************************
*                                                                       *
*                                                                       *
*                                                                       *
*        READ ROUTINE FOR DIRECT ACCESS FILE I/O                        *
*                                                                       *
*                                                                       *
*        INPUT TO THIS ROUTINE IS:                                      *
*                                                                       *
*      PARM1   CORE ADDRESS TO READ THE RECORD INTO                     *
*                                                                       *
*      SVCODE  SERVICE CODE INDICATING WHICH FILE TO USE                *
*                                                                       *
*      PARM2   RELATIVE RECORD NUMBER   0,1,2,3,...                     *
*                                                                       *
*                                                                       *
*                                                                       *
*************************************************************************
         SPACE 2
READ     DS    0H
         ST    PARM1,RDECB+12     STORE ADDRESS
         L     3,ARWDCBS-FILEORG(SVCODE)
*                                 ADDRESS OF THE DCB FOR THIS FILE
         USING IHADCB,3
         TM    DCBOFLGS,OPENBIT   IS THE FILE OPEN ?
```

```
         BO    READOPEN            YES, GO READ
         ST    3,OCDCB             STORE DCB ADDRESS FOR OPEN SVC
         MVI   OCDCB,X'80'         FLAG END OF PARAMETER LIST
*                                  AND INDICATE OPEN FOR INPUT
         OPEN  ,MF=(E,OCDCB)       OPEN THE FILE
         TM    DCBOFLGS,OPENBIT    WAS THE OPEN SUCCESSFUL ?
         BZ    FILESYND            NO, SYNAD ERROR
         SPACE 1
READOPEN DS    0H
         TM    DCBDEVT,TAPEBITS    IS THE FILE ON MAGNETIC TAPE
         DROP  3
         BO    READTP              YES, GO FORM RECORD INDEX FOR TAPE
         SLA   PARM2,16            FORM  TTRZ  ADDRESS
         BNZ   RDNO                BLOCK ZERO IS A SPECIAL CASE
         LA    PARM2,1             FUNNY ADDRESS FOR BLOCK ZERO
         B     READTP              GO DO THE READ
RDNO     O     PARM2,TTRSET        SPECIFY LOGICAL RECORD 1
READTP   ST    PARM2,TTR           SAVE RECORD POINTER
         SPACE 1
         POINT (3),TTR             POINT AT THE RECORD TO BE READ
         READ  RDECB,SF,(3),0,'S'  READ THE RECORD INTO CORE
         CHECK RDECB               WAIT FOR THE READ TO COMPLETE
         SPACE 1
         B     EXIT                RETURN TO THE XPL PROGRAM
         EJECT
         SPACE 5
*********************************************************************
*                                                                   *
*                                                                   *
*                                                                   *
*        WRITE ROUTINE FOR DIRECT ACCESS FILE I/O                   *
*                                                                   *
*                                                                   *
*        INPUT TO THIS ROUTINE IS:                                  *
*                                                                   *
*      PARM1   CORE ADDRESS TO READ THE RECORD FROM                 *
*                                                                   *
*      SVCODE  SERVICE CODE INDICATING WHICH FILE TO USE            *
*                                                                   *
*      PARM2   RELATIVE RECORD NUMBER   0,1,2, ...                  *
*                                                                   *
*                                                                   *
*                                                                   *
*********************************************************************
         SPACE 2
WRITE    DS    0H
         ST    PARM1,WDECB+12      SAVE CORE ADDRESS
         L     3,ARWDCBS-FILEORG(SVCODE)
*                                  GET THE DCB ADDRESS
         USING IHADCB,3
         TM    DCBOFLGS,OPENBIT    IS THE FILE OPEN ?
         BO    WRTOPEN             YES, GO WRITE
         ST    3,OCDCB             STORE DCB ADDRESS FOR OPEN SVC
         MVI   OCDCB,X'8F'         FLAG END OF ARGUMENT LIST AND
*                                  INDICATE OPENING FOR OUTPUT
         SPACE 1
```

```
         OPEN  ,MF=(E,OCDCB)        OPEN THE FILE
         SPACE 1
         TM    DCBOFLGS,OPENBIT     WAS THE OPEN SUCCESSFUL ?
         BZ    FILESYND             NO,SYNAD ERROR
         SPACE 1
WRTOPEN  DS    0H
         TM    DCBDEVT,TAPEBITS     IS THE FILE ON MAGNETIC TAPE
         DROP  3
         BO    WRITP                YES, GO FORM RECORD INDEX FOR TAPE
         SLA   PARM2,16             FORM TTRZ ADDRESS FOR DIRECT ACCESS
         BNZ   WRDNO                RECORD ZERO IS A SPECIAL CASE
         LA    PARM2,1              FUNNY ADDRESS FOR RECORD ZERO
         B     WRITP                GO DO THE WRITE
WRDNO    O     PARM2,TTRSET         OR IN RECORD NUMBER BIT
WRITP    ST    PARM2,TTR            SAVE RECORD POINTER
         SPACE 1
         POINT (3),TTR              POINT AT THE DESIRED RECORD
         WRITE WDECB,SF,(3),0,'S'   WRITE THE RECORD OUT
         CHECK WDECB                WAIT FOR THE WRITE TO FINISH
         SPACE 1
         B     EXIT                 RETURN TO THE XPL PROGRAM
         EJECT
         SPACE 5
*****************************************************************************
*                                                                           *
*                                                                           *
*                                                                           *
*        TIME AND DATE FUNCTIONS                                            *
*                                                                           *
*                                                                           *
*        RETURNS TIME OF DAY IN HUNDREDTHS OF A SECOND IN REGISTER          *
*        PARM1  AND THE DATE IN THE FORM  YYDDD IN REGISTER SVCODE          *
*                                                                           *
*                                                                           *
*****************************************************************************
         SPACE 2
GETIME   TIME  BIN                  REQUEST THE TIME
         ST    0,SAVREG+PARM1*4     RETURN IN REGISTER PARM1
         ST    1,DTSV+4             STORE THE DATE IN PACKED DECIMAL
         CVB   1,DTSV               CONVERT IT TO BINARY
         ST    1,SAVREG+SVCODE*4    RETURN DATE IN REGISTER SVCODE
         B     EXIT                 RETURN TO THE XPL PROGRAM
         EJECT
         SPACE 5
*****************************************************************************
*                                                                           *
*                                                                           *
*                                                                           *
*        DATA AREA FOR THE SUBMONITOR                                       *
*                                                                           *
*                                                                           *
*****************************************************************************
         SPACE 2
         DS    0F
         DC    A(SAVE)              ADDRESS OF OS SAVE AREA
         DC    A(HIGHSERV-4)        LARGEST VALID SERVICE CODE
```

```
RTNSV    DC    F'0'                SAVE COMPLETION CODE RETURNED
*                                  BY THE XPL PROGRAM
ABESAVE  DS    F                   SAVE ABEND CODE DURING CLOSE
ABEREGS  DS    3F                  SAVE PROGRAMS REGS 0-2 BEFORE ABEND
TTR      DC    F'0'                TTRZ ADDRESS FOR READ AND WRITE
TTRSET   DC    X'00000100'         ADDRESS CONSTANT FOR TTRZ
FLAGS    DC    X'00'               SUBMONITOR CONTROL FLAGS
SAVREG   DC    16F'0'              SAVE AREA FOR THE SUBMONITOR
         DS    0D
DTSV     DC    PL8'0'              WORK AREA FOR CONVERTING DATE
         SPACE 2
*********************************************************************
*                                                                   *
*                                                                   *
*        DCB ADDRESS TABLE FOR ALL I/O ROUTINES                     *
*                                                                   *
*                                                                   *
*        THE FOUR SETS OF DCB ADDRESSES HEADED BY  'GETDCBS',       *
*        'PUTDCBS', 'ARWDCBS', AND 'PGMDCB' MUST BE CONTIGUOUS      *
*        AND END WITH 'PGMDCB'.  THESE LISTS ARE USED AT JOB END    *
*        TO CLOSE ALL FILES BEFORE RETURNING TO OS                  *
*                                                                   *
*                                                                   *
*        DCB ADDRESSES FOR INPUT FILES:                             *
*                                                                   *
*                                                                   *
*********************************************************************
         SPACE 2
         PRINT NOGEN
         SPACE 1
GETDCBS  DS    0F
&I       SETA  0
.GD1     AIF   (&I GT &INPUTS).GD2
         DC    A(INPUT&I)
&I       SETA  &I+1
         AGO   .GD1
.GD2     ANOP
         SPACE 1
*********************************************************************
*                                                                   *
*        DCB ADDRESSES FOR OUTPUT FILES                             *
*                                                                   *
*********************************************************************
         SPACE 1
PUTDCBS  DS    0F
         SPACE 1
&I       SETA  0
.PD1     AIF   (&I GT &OUTPUTS).PD2
         DC    A(OUTPUT&I)
&I       SETA  &I+1
         AGO   .PD1
.PD2     ANOP
         SPACE 1
*********************************************************************
*                                                                   *
*        DCB ADDRESS FOR DIRECT ACCESS FILES                        *
```

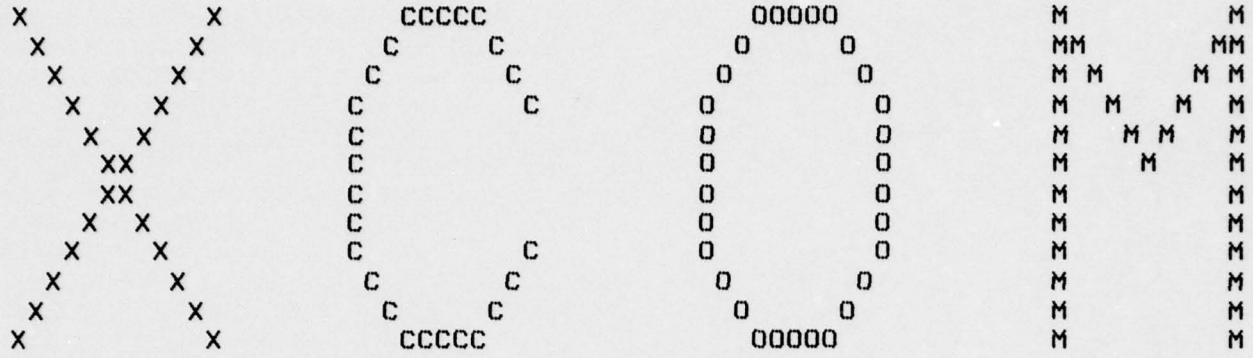
```
*                                                                      *
************************************************************************
          SPACE 1
ARWDCBS   DS    0F
          SPACE 1
&I        SETA  1
.DA1      AIF   (&I GT &FILES).DA2
          ORG   ARWDCBS+RD&I-FILEORG
          DC    A(FILE&I.IN)
          ORG   ARWDCBS+WRT&I-FILEORG
          DC    A(FILE&I.OUT)
&I        SETA  &I+1
          AGO   .DA1
.DA2      ANOP
          ORG   ARWDCBS+ENDSERV-FILEORG
          DS    0F
          SPACE 2
PGMDCB    DC    X'80'               FLAG END OF PARAMETER LIST
          DC    AL3(PROGRAM)        ADDRESS OF PROGRAM DCB
          SPACE 2
OCDCB     DS    F                   DCB ADDRESSES FOR OPEN AND CLOSE
MOVEADR   DS    1F                  DESCRIPTOR STORAGE FOR PUT ROUTINE
FILLENG   DC    F'0'                LENGTH OF PADDING NEEDED IN OUTPUT
F1        DC    F'1'                THE CONSTANT ONE
F2        DC    F'2'                THE CONSTANT TWO
GETMOVE   MVC   0(0,2),0(1)         MVC COMMAND FOR THE GET ROUTINE
MVCNULL   MVC   1(0,1),0(1)         MVC COMMAND FOR THE PUT ROUTINE
MVCBLANK  MVC   2(0,1),1(1)              .
MVCSTRNG  MVC   0(0,1),0(2)              .
MVCPO     MVC   1(0,1),0(2)              .
MVCPOBL   MVC   3(0,1),2(1)              .
          SPACE 1
************************************************************************
*                                                                      *
*         DATA AREA FOR THE EXECUTE ROUTINE                            *
*                                                                      *
************************************************************************
          SPACE 1
XCELL     DS    1F                  INSTRUCTION TO BE EXECUTED
          DS    1F                  MORE INSTRUCTION
          DS    1F                  ADDRESS OF TRACE ROUTINE REGISTER
*                                   TABLE
          DS    1F                  RETURN ADDRESS TO THE TRACE ROUTINE
*                                   ADDRESS OF EXEC IS IN REGISTER FOUR
EXSV      DS    2F                  WORK AREA FOR THE EXECUTE ROUTINE
          EJECT
          SPACE 5
************************************************************************
*                                                                      *
*                                                                      *
*                                                                      *
*         DEVICE  CONTROL  BLOCKS  FOR  THE  SUBMONITOR                *
*                                                                      *
*                                                                      *
************************************************************************
          SPACE 2
```

```
PROGRAM  DCB    DSORG=PS,                                              X
                MACRF=R,                                               X
                DDNAME=PROGRAM,                                        X
                DEVD=DA,                                               X
                KEYLEN=0,                                              X
                EODAD=EODPGM,                                          X
                SYNAD=ERRPGM
         SPACE 2
INPUT0   DCB    DSORG=PS,                                              X
                DDNAME=SYSIN,                                          X
                DEVD=DA,                                               X
                MACRF=GL,                                              X
                BUFNO=3,                                               X
                EODAD=INEOD,                                           X
                SYNAD=INSYNAD,                                         X
                EXLST=INEXIT0,                                         X
                EROPT=ACC
         SPACE 1
*****************************************************************************
*                                                                           *
INPUT1   EQU    INPUT0              INPUT(0) & INPUT(1) ARE BOTH SYSIN *
*                                                                           *
*****************************************************************************
         SPACE 2
&I       SETA   2
.INP1    AIF    (&I GT &INPUTS).INP2
         SPACE 1
INPUT&I  DCB    DSORG=PS,                                              X
                DDNAME=INPUT&I,                                        X
                DEVD=DA,                                               X
                MACRF=GL,                                              X
                EODAD=INEOD,                                           X
                SYNAD=INSYNAD,                                         X
                EXLST=INEXIT&I,                                        X
                EROPT=ACC
         SPACE 1
&I       SETA   &I+1
         AGO    .INP1
.INP2    ANOP
         SPACE 2
OUTPUT0  DCB    DSORG=PS,                                              X
                DDNAME=SYSPRINT,                                       X
                DEVD=DA,                                               X
                MACRF=PL,                                              X
                SYNAD=OUTSYNAD,                                        X
                EXLST=OUTEXIT0,                                        X
                EROPT=ACC
         SPACE 1
*****************************************************************************
*                                                                           *
OUTPUT1  EQU    OUTPUT0             OUTPUT(0), OUTPUT(1) BOTH SYSPRINT *
*                                                                           *
*****************************************************************************
         SPACE 2
OUTPUT2  DCB    DSORG=PS,                                              X
                DDNAME=SYSPUNCH,                                       X
```

```
                DEVD=DA,                                                X
                MACRF=PL,                                               X
                SYNAD=OUTSYNAD,                                         X
                EXLST=OUTEXIT2,                                         X
                EROPT=ACC
          SPACE 1
&I        SETA  3
.OP1      AIF   (&I GT &OUTPUTS).OP2
          SPACE 1
OUTPUT&I  DCB   DSORG=PS,                                               X
                DDNAME=OUTPUT&I,                                        X
                DEVD=DA,                                                X
                MACRF=PL,                                               X
                SYNAD=OUTSYNAD,                                         X
                EXLST=OUTEXIT&I,                                        X
                EROPT=ACC
          SPACE 1
&I        SETA  &I+1
          AGO   .OP1
.OP2      ANOP
          SPACE 1
*****************************************************************************
*                                                                           *
*                                                                           *
*         DCBS FOR THE DIRECT ACCESS FILES                                  *
*                                                                           *
*                                                                           *
*         BECAUSE OF THE MANNER IN WHICH THE FILES ARE USED,  IT IS         *
*         NECESSARY TO HAVE TWO DCB'S FOR EACH FILE.  ONE DCB FOR           *
*         READING AND ONE FOR WRITING.                                      *
*                                                                           *
*                                                                           *
*****************************************************************************
          SPACE 2
&I        SETA  1
.DD1      AIF   (&I GT &FILES).DD2
          SPACE 1
FILE&I.IN DCB   DSORG=PS,                                               X
                MACRF=RP,                                               X
                DDNAME=FILE&I,                                          X
                DEVD=DA,                                                X
                RECFM=F,                                                X
                LRECL=FILEBYTS,                                         X
                BLKSIZE=FILEBYTS,                                       X
                KEYLEN=0,                                               X
                EODAD=FILEEOD,                                          X
                SYNAD=FILESYND
          SPACE 2
FILE&I.OUT DCB  DSORG=PS,                                               X
                MACRF=WP,                                               X
                DDNAME=FILE&I,                                          X
                DEVD=DA,                                                X
                RECFM=F,                                                X
                KEYLEN=0,                                               X
                LRECL=FILEBYTS,                                         X
                BLKSIZE=FILEBYTS,                                       X
```

```
                SYNAD=FILESYND
          SPACE 1
&I        SETA  &I+1
          AGO   .DD1
.DD2      ANOP
          SPACE 4
XPLSMEND  DS    0H                    END  OF  THE  SUBMONITOR
          EJECT
          SPACE 5
*****************************************************************************
*                                                                           *
*                                                                           *
*         DUMMY  DCB  FOR  DEFINING  DCB  FIELDS                            *
*                                                                           *
*                                                                           *
*****************************************************************************
          SPACE 2
          DCBD  DSORG=QS,DEVD=DA
          EJECT
          SPACE 5
*****************************************************************************
*                                                                           *
*                                                                           *
*         THE  END                                                          *
*                                                                           *
*                                                                           *
*****************************************************************************
          SPACE 5
          END
```

/*



THE COMPILER FOR X P L

W. M. MCKEEMAN	J. J. HORNING	D. B. WORTMAN
INFORMATION & COMPUTER SCIENCE,	COMPUTER SCIENCE DEPARTMENT,	COMPUTER SCIENCE DEPARTMENT,
UNIVERSITY OF CALIFORNIA AT	STANFORD UNIVERSITY,	STANFORD UNIVERSITY,
SANTA CRUZ, CALIFORNIA 95060	STANFORD, CALIFORNIA 94305	STANFORD, CALIFORNIA 94305

DEVELOPED AT THE STANFORD COMPUTATION CENTER, CAMPUS FACILITY, 1966-69
AND THE UNIVERSITY OF CALIFORNIA COMPUTATION CENTER, SANTA CRUZ, 1968-69.

MODIFIED AT WEST VIRGINIA UNIVERSITY, MORGANTOWN WEST VIRGINIA 1978
MODIFICATIONS INCLUDE :

1. GENERATION OF LINKAGE EDITABLE OBJECT CODE
2. PSEUDO SYNTAX FOR EXTERNAL AND PUBLIC MODULES
3. PSEUDO SYNTAX FOR COMMON DATA VARIABLES
4. THE XPL SUBMONITOR AND COMPACTIFY ROUTINES ARE NOW GENERALIZED SUBROUTINES EXTERNAL TO THE MAIN PROGRAM MODULES
6. STRING ASSIGNMENT IS MODIFIED TO ENSURE UNIQUE COPIES OF ANY CHARACTER STRING, THEREBY ELIMINATING THE PROBLEM OF INADVERTENT MODIFICATION OF SHARED DESCRIPTORS
7. INCLUDES GREATER USE OF CHARACTER CONSTANTS AS VARIABLES TO REDUCE THE NUMBER OF DESCRIPTORS REQUIRED WITHIN THE THE XCOM COMPILER
8. UNFORTUNATELY THE "TRACE" CAPABILITY HAS BEEN LOST BECAUSE OF THE REGISTER REASSIGNMENT PERFORMED (I.E. R15 NO LONGER POINTS TO THE SUBMONITOR ENTRY EXCLUSIVELY)
9. AS WAS JUST MENTIONED, THE REGISTER ASSIGNMENTS HAVE BEEN CHANGED TO CONFORM TO STANDARD IBM USAGE (I.E. R14 IS THE LINKAGE RETURN REG R15 IS THE LINKAGE CALL REGISTER, ETC...)
10. REGISTERS ARE SAVE VIA IBM CONVENTION UPON ENTRY AND EXIT FROM ALL "PUBLIC" MODULES AND THE MAIN MODULE

THE MODIFICATIONS WERE INSTALLED BY CAPT. GARY R. FELDBAUER, USAF
DURING THE SUMMER OF 1978 AS A MASTERS PROJECT IN COMPUTER SCIENCE.

```
                                                                    */

/*  FIRST WE INITIALIZE THE GLOBAL CONSTANTS THAT DEPEND UPON THE INPUT
    GRAMMAR.  THE FOLLOWING CARDS ARE PUNCHED BY THE SYNTAX PRE-PROCESSOR  */

DECLARE NSY LITERALLY '91', NT LITERALLY '42';
DECLARE V(NSY) CHARACTER INITIAL ('<ERROR: TOKEN = 0>', ';', ')', '(', ',',
    ':', '=', '\', '&', '^', '<', '>', '+', '-', '*', '/', 'IF', 'DO', 'TO',
    'BY', 'GO', '\\', '_\_', 'END', 'BIT', 'MOD', 'THEN', 'ELSE', 'CASE',
    'CALL', 'GOTO', 'WHILE', 'FIXED', 'LABEL', 'RETURN', 'DECLARE', 'INITIAL',
    '<STRING>', '<NUMBER>', 'PROCEDURE', 'LITERALLY', 'CHARACTER',
    '<IDENTIFIER>', '<TYPE>', '<TERM>', '<GROUP>', '<GO TO>', '<ENDING>',
    '<PROGRAM>', '<REPLACE>', '<PRIMARY>', '<VARIABLE>', '<BIT HEAD>',
    '<CONSTANT>', '<RELATION>', '<STATEMENT>', '<IF CLAUSE>', '<TRUE PART>',
    '<LEFT PART>', '<ASSIGNMENT>', '<EXPRESSION>', '<GROUP HEAD>',
    '<BOUND HEAD>', '<IF STATEMENT>', '<WHILE CLAUSE>', '<INITIAL LIST>',
    '<INITIAL HEAD>', '<CASE SELECTOR>', '<STATEMENT LIST>',
    '<CALL STATEMENT>', '<PROCEDURE HEAD>', '<PROCEDURE NAME>',
    '<PARAMETER LIST>', '<PARAMETER HEAD>', '<LOGICAL FACTOR>',
    '<SUBSCRIPT HEAD>', '<BASIC STATEMENT>', '<GO TO STATEMENT>',
    '<STEP DEFINITION>', '<IDENTIFIER LIST>', '<LOGICAL PRIMARY>',
    '<RETURN STATEMENT>', '<LABEL DEFINITION>', '<TYPE DECLARATION>',
    '<ITERATION CONTROL>', '<LOGICAL SECONDARY>', '<STRING EXPRESSION>',
    '<DECLARATION ELEMENT>', '<PROCEDURE DEFINITION>',
    '<DECLARATION STATEMENT>', '<ARITHMETIC EXPRESSION>',
    '<IDENTIFIER SPECIFICATION>');
DECLARE V_INDEX(12) FIXED INITIAL (1, 16, 22, 26, 31, 34, 35, 37, 39, 42,
    42, 42, 43);
DECLARE C1(NSY) BIT(86) INITIAL (
    "(2) 00000 00000 00000 00000 00000 00000 00000 00000 000",
    "(2) 02000 00000 00000 02200 20220 00202 20002 20000 002",
    "(2) 02222 02222 22222 20022 02003 22000 00330 02000 030",
    "(2) 00030 00003 00330 00000 00000 00000 00000 00330 003",
    "(2) 00030 00002 00220 00000 00000 00000 00000 00220 003",
    "(2) 02000 00000 00000 02200 20020 00002 20002 20002 002",
    "(2) 00020 00002 00220 00000 00000 00000 00000 00220 002",
    "(2) 00010 00001 00110 00000 00000 00000 00000 00110 001",
    "(2) 00010 00001 00110 00000 00000 00000 00000 00110 001",
    "(2) 00010 01000 11110 00000 00000 00000 00000 00110 001",
    "(2) 00020 01000 00220 00000 00000 00000 00000 00220 002",
    "(2) 00020 01000 00220 00000 00000 00000 00000 00220 002",
    "(2) 00010 00000 00000 00000 00000 00000 00000 00110 001",
    "(2) 00010 00000 00000 00000 00000 00000 00000 00110 001",
    "(2) 00010 00000 00000 00000 00000 00000 00000 00110 001",
    "(2) 00010 00000 00000 00000 00000 00000 00000 00110 001",
    "(2) 00010 00001 00110 00000 00000 00000 00000 00110 001",
    "(2) 01000 00000 00000 00000 00000 00010 01000 00000 001",
    "(2) 00010 00001 00110 00000 00000 00000 00000 00110 003",
    "(2) 00010 00001 00110 00000 00000 00000 00000 00110 001",
    "(2) 00000 00000 00000 00010 00000 00000 00000 00000 000",
    "(2) 00010 00000 00110 00000 00000 00000 00000 00110 001",
    "(2) 01000 00000 00000 01100 10000 00001 10001 10000 001",
    "(2) 02000 00000 00000 00000 00000 00000 00000 00000 001",
```

```
"(2) 00010 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 00010 00000 00000 00000 00000 00000 00000 00110 001",
"(2) 02000 00000 00000 02200 20000 00002 20002 20000 002",
"(2) 02000 00000 00000 02200 20000 00002 20002 20000 002",
"(2) 00010 00001 00110 00000 00000 00000 00000 00110 001",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 001",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 002",
"(2) 00010 00001 00110 00000 00000 00000 00000 00110 001",
"(2) 02002 00000 00000 00000 00000 00000 00000 02000 000",
"(2) 02002 00000 00000 00000 00000 00000 00000 02000 000",
"(2) 02010 00001 00110 00000 00000 00000 00000 00110 001",
"(2) 00010 00000 00000 00000 00000 00000 00000 00000 001",
"(2) 00010 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 02202 02222 22222 20022 02000 22000 00000 00000 000",
"(2) 02302 02222 22222 20022 02000 22000 00000 00000 000",
"(2) 02020 00000 00000 00000 00002 00000 00220 00000 020",
"(2) 00000 00000 00000 00000 00000 00000 00000 00100 000",
"(2) 02002 00000 00000 00000 00000 00000 00000 02000 000",
"(2) 02333 12222 22222 20022 02002 22000 00220 00000 120",
"(2) 03002 00000 00000 00000 00000 00000 00000 02000 000",
"(2) 02202 02222 22221 10022 02000 12000 00000 00000 000",
"(2) 01000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 001",
"(2) 02000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 00010 00001 00110 00000 00000 00000 00000 00110 001",
"(2) 02202 02222 22222 20022 02000 22000 00000 00000 000",
"(2) 02203 03222 22222 20022 02000 22000 00000 00000 000",
"(2) 00000 00000 00000 00000 00000 00000 00000 00010 000",
"(2) 02303 02222 22222 20022 02000 22000 00000 00000 000",
"(2) 00010 00000 00110 00000 00000 00000 00000 00110 001",
"(2) 02000 00000 00000 02200 20220 00002 20002 20000 002",
"(2) 01000 00000 00000 01100 10000 00001 10001 10000 001",
"(2) 01000 00000 00000 01100 10000 00001 10001 10000 001",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 001",
"(2) 03000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 02101 00160 00000 00011 00000 01000 00000 00000 000",
"(2) 01000 00000 00000 01100 10010 00001 10001 10000 001",
"(2) 00000 00000 00000 00000 00000 00000 00000 00010 000",
"(2) 02000 00000 00000 02200 20220 00002 20002 20000 002",
"(2) 01000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 02002 00000 00000 00000 00000 00000 00000 02000 000",
"(2) 00000 00000 00000 00000 00000 00000 00000 00110 000",
"(2) 01000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 01000 00000 00000 01100 10210 00001 10001 10000 001",
"(2) 01000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 01000 00000 00000 01100 10000 00001 10001 10000 001",
"(2) 01010 00000 00000 00000 00001 00000 00110 00000 010",
"(2) 01000 00000 00000 00000 00001 00000 00110 00000 010",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 001",
"(2) 02202 00210 00000 00022 00000 02000 00000 00000 000",
"(2) 00010 00001 00110 00000 00000 00000 00000 00110 001",
"(2) 02000 00000 00000 02200 20220 00302 20002 20000 002",
"(2) 01000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 01000 00000 00000 00000 00000 00000 00000 00000 000",
"(2) 00000 00000 00000 00000 00000 00000 00000 00000 001",
```

```
    '(2) 02202 00220 00000 00022 00000 02000 00000 00000 000',
    '(2) 01000 00000 00000 00000 00000 00000 00000 00000 000',
    '(2) 01000 00000 00000 01100 10010 00001 10001 10001 001',
    '(2) 02002 00000 00000 00000 00000 00000 00000 01000 000',
    '(2) 02000 00000 00000 00000 00000 00000 00000 00000 000',
    '(2) 02202 00220 00000 00022 00000 02000 00000 00000 000',
    '(2) 02202 01221 11000 00022 01000 02000 00000 00000 000',
    '(2) 02002 00000 00000 00000 00000 00000 00000 00000 000',
    '(2) 01000 00000 00000 00000 00000 00000 00000 00000 000',
    '(2) 01001 00000 00000 00000 00000 00000 00000 00000 000',
    '(2) 02202 02222 22110 00022 02000 02000 00000 00000 000',
    '(2) 00010 00000 00000 00000 00001 00000 00110 00000 010');
DECLARE NC1TRIPLES LITERALLY '203';
DECLARE C1TRIPLES(NC1TRIPLES) FIXED INITIAL (197379, 197385, 197388, 197389,
   197413, 197414, 197418, 207363, 459523, 459529, 459532, 459533, 459557,
   459558, 459562, 469507, 525059, 525065, 525068, 525069, 525093, 525094,
   525098, 535043, 590595, 590601, 590604, 590605, 590629, 590630, 590634,
   600579, 787203, 787209, 787212, 787213, 787237, 787238, 787242, 797187,
   852739, 852745, 852748, 852749, 852773, 852774, 852778, 862723, 918275,
   918281, 918284, 918285, 918309, 918310, 918314, 928259, 983811, 983817,
   983820, 983821, 983845, 983846, 983850, 993795, 1049347, 1049353, 1049356,
   1049357, 1049381, 1049382, 1049386, 1059331, 1124867, 1127174, 1180419,
   1180425, 1180428, 1180429, 1180453, 1180454, 1180458, 1190403, 1245955,
   1245961, 1245964, 1245965, 1245989, 1245990, 1245994, 1255939, 1377027,
   1377033, 1377036, 1377037, 1377061, 1377062, 1377066, 1387011, 1452547,
   1454852, 1454854, 1456897, 1639171, 1639177, 1639180, 1639181, 1639205,
   1639206, 1639210, 1649155, 1835779, 1835785, 1835788, 1835789, 1835813,
   1835814, 1835818, 1845763, 1911299, 2032387, 2032393, 2032396, 2032397,
   2032421, 2032422, 2032426, 2042371, 2228995, 2229001, 2229004, 2229005,
   2229029, 2229030, 2229034, 2238979, 2490904, 2490912, 2490913, 2490921,
   3212035, 3212041, 3212044, 3212045, 3212069, 3212070, 3212074, 3222019,
   3417602, 3539715, 3539721, 3539724, 3539725, 3539749, 3539750, 3539754,
   3549699, 3680771, 3683076, 3683078, 3685121, 3689499, 3746307, 3748612,
   3748614, 3750657, 3811843, 3814148, 3814150, 3936810, 4008451, 4010756,
   4010758, 4012801, 4072962, 4338946, 4338948, 4467203, 4469508, 4469510,
   4471553, 4598275, 4600580, 4600582, 4602625, 4664065, 4729601, 4794882,
   4794884, 4915971, 4915977, 4915980, 4915981, 4916005, 4916006, 4916010,
   4925955, 5188098, 5188100, 5384707, 5387012, 5387014, 5389057, 5833731,
   5833770);
DECLARE PRTB(109) FIXED INITIAL (0, 4671531, 18219, 18248, 4430, 4416, 4419,
   71, 17, 59, 45, 88, 81, 69, 77, 89, 0, 16949, 18730, 13350, 20266, 828,
   19260, 91, 36, 24, 42, 0, 0, 18730, 20266, 16949, 19260, 51, 42, 9, 10,
   11, 0, 0, 9, 0, 9, 0, 20, 0, 4156, 76, 0, 0, 0, 0, 10792, 0, 0, 82, 0, 23,
   46, 0, 0, 4072962, 91, 23052, 23053, 12, 13, 0, 17988, 82, 61, 11278,
   11279, 11289, 0, 29, 0, 0, 14393, 68, 61, 56, 0, 58, 1195027, 13105, 18,
   31, 28, 34, 82, 0, 83, 0, 15367, 0, 82, 0, 9, 0, 0, 3354940, 18952, 0,
   22070, 0, 22788, 35, 22037, 0);
DECLARE PRDTB(109) BIT(8) INITIAL (0, 35, 33, 34, 22, 23, 24, 32, 21, 6, 7,
   8, 9, 10, 11, 12, 13, 67, 37, 60, 64, 103, 105, 62, 68, 61, 106, 38, 65,
   39, 66, 69, 107, 73, 43, 85, 88, 89, 82, 72, 86, 83, 87, 84, 48, 40, 18,
   19, 49, 57, 59, 44, 53, 108, 109, 36, 58, 41, 47, 63, 104, 55, 54, 93, 94,
   95, 96, 92, 31, 42, 20, 98, 99, 100, 97, 46, 102, 101, 16, 3, 25, 15, 2,
   71, 28, 70, 27, 29, 30, 45, 17, 5, 56, 1, 75, 74, 14, 4, 79, 78, 52, 26,
   77, 76, 81, 80, 51, 50, 91, 90);
DECLARE HDTB(109) BIT(8) INITIAL (0, 70, 70, 70, 61, 61, 61, 70, 61, 76, 76,
   76, 76, 76, 76, 76, 76, 65, 72, 43, 91, 50, 51, 62, 66, 52, 75, 73, 79,
```

```
   73, 79, 66, 75, 58, 82, 54, 54, 54, 54, 49, 54, 54, 54, 54, 46, 47, 56,
   57, 46, 43, 43, 81, 87, 53, 53, 71, 43, 47, 77, 91, 51, 83, 83, 90, 90,
   90, 90, 90, 88, 47, 45, 44, 44, 44, 44, 69, 50, 50, 63, 68, 61, 63, 68,
   59, 84, 59, 84, 64, 67, 81, 63, 55, 83, 48, 60, 60, 76, 55, 85, 85, 87,
   78, 74, 74, 80, 80, 89, 89, 86, 86);
DECLARE PRLENGTH(109) BIT(8) INITIAL (0, 4, 3, 3, 3, 3, 3, 2, 2, 2, 2, 2, 2,
   2, 2, 2, 2, 1, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2, 1, 1, 3, 3, 3, 3, 2, 2, 2, 2, 2,
   1, 1, 2, 1, 2, 1, 2, 1, 3, 2, 1, 1, 1, 1, 3, 1, 1, 2, 1, 2, 2, 1, 1, 4, 2,
   3, 3, 2, 2, 1, 3, 2, 2, 3, 3, 3, 1, 2, 1, 1, 3, 2, 2, 2, 1, 2, 4, 3, 2, 2,
   2, 2, 2, 1, 2, 1, 3, 1, 2, 1, 2, 1, 1, 4, 3, 1, 3, 1, 3, 2, 3, 1);
DECLARE CONTEXT_CASE(109) BIT(8) INITIAL (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0);
DECLARE LEFT_CONTEXT(1) BIT(8) INITIAL (86, 71);
DECLARE LEFT_INDEX(49) BIT(8) INITIAL (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2,
   2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2);
DECLARE CONTEXT_TRIPLE(0) FIXED INITIAL (0);
DECLARE TRIPLE_INDEX(49) BIT(8) INITIAL (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
   0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1);
DECLARE PR_INDEX(91) BIT(8) INITIAL (1, 17, 23, 29, 34, 35, 40, 40, 40, 40,
   42, 44, 44, 44, 44, 44, 44, 44, 45, 45, 45, 45, 45, 46, 46, 46, 47, 48,
   48, 48, 49, 49, 50, 51, 52, 52, 52, 54, 55, 56, 56, 57, 61, 63, 68, 68,
   68, 71, 71, 71, 75, 77, 77, 78, 78, 83, 83, 83, 83, 84, 90, 90, 90, 92,
   92, 93, 93, 93, 94, 94, 94, 94, 94, 94, 96, 96, 98, 98, 98, 98, 100, 100,
   100, 101, 102, 104, 106, 108, 108, 108, 110, 110);

/*  END OF CARDS PUNCHED BY SYNTAX                                          */

/* DECLARES FOR BINARY (LINK EDIT) DATA SET CREATION */

DECLARE SYSPUNCH LITERALLY 'OUTPUT(2)';
DECLARE BINRCD CHARACTER;
DECLARE BINARY(79) BIT(8);  /* BINARY CODE RECORD */
DECLARE BINPTR FIXED;

/*  DECLARATIONS FOR THE SCANNER                                            */

/* TOKEN IS THE INDEX INTO THE VOCABULARY V() OF THE LAST SYMBOL SCANNED,
   CH IS THE LAST CHARACTER SCANNED (HEX CODE),
   CP IS THE POINTER TO THE LAST CHARACTER SCANNED IN THE CARDIMAGE,
   BCD IS THE LAST SYMBOL SCANNED (LITERAL CHARACTER STRING). */

DECLARE (TOKEN, CH, CP) FIXED, BCD CHARACTER;

/* SET UP SOME CONVENIENT ABBREVIATIONS FOR PRINTER CONTROL */
DECLARE EJECT_PAGE LITERALLY 'OUTPUT(1) = PAGE',
   PAGE CHARACTER INITIAL ('1'), DOUBLE CHARACTER INITIAL ('0'),
   DOUBLE_SPACE LITERALLY 'OUTPUT(1) = DOUBLE',
   X70 CHARACTER INITIAL ('
                ');

/* LENGTH OF LONGEST SYMBOL IN V */
```

```
DECLARE (RESERVED_LIMIT, MARGIN_CHOP) FIXED;

/* CHARTYPE() IS USED TO DISTINGUISH CLASSES OF SYMBOLS IN THE SCANNER.
   TX() IS A TABLE USED FOR TRANSLATING FROM ONE CHARACTER SET TO ANOTHER.
   CONTROL() HOLDS THE VALUE OF THE COMPILER CONTROL TOGGLES SET IN $ CARDS.
   NOT_LETTER_OR_DIGIT() IS SIMILIAR TO CHARTYPE() BUT USED IN SCANNING
   IDENTIFIERS ONLY.

   ALL ARE USED BY THE SCANNER AND CONTROL() IS SET THERE.
*/
DECLARE (CHARTYPE, TX) (255) BIT(8),
        (CONTROL, NOT_LETTER_OR_DIGIT)(255) BIT(1);

/* ALPHABET CONSISTS OF THE SYMBOLS CONSIDERED ALPHABETIC IN BUILDING
   IDENTIFIERS     */
DECLARE ALPHABET CHARACTER INITIAL ('ABCDEFGHIJKLMNOPQRSTUVWXYZ_$@#');

/* BUFFER HOLDS THE LATEST CARDIMAGE,
   TEXT HOLDS THE PRESENT STATE OF THE INPUT TEXT (INCLUDING MACRO
   EXPANSIONS AND NOT INCLUDING THE PORTIONS DELETED BY THE SCANNER),
   TEXT_LIMIT IS A CONVENIENT PLACE TO STORE THE POINTER TO THE END OF TEXT,
   CARD_COUNT IS INCREMENTED BY ONE FOR EVERY XPL SOURCE CARD READ,
   ERROR_COUNT TABULATES THE ERRORS AS THEY ARE DETECTED DURING COMPILE,
   SEVERE_ERRORS TABULATES THOSE ERRORS OF FATAL SIGNIFICANCE.
*/
DECLARE (BUFFER, TEXT, CURRENT_PROCEDURE, INFORMATION) CHARACTER,
   (TEXT_LIMIT, CARD_COUNT, ERROR_COUNT, SEVERE_ERRORS, PREVIOUS_ERROR) FIXED
   ;

/* NUMBER_VALUE CONTAINS THE NUMERIC VALUE OF THE LAST CONSTANT SCANNED,
   JBASE CONTAINS THE FIELD WIDTH IN BIT STRINGS (DEFAULT VALUE = 4),
   BASE IS  2**JBASE   (I.E., SHL(1,JBASE) ).
*/
DECLARE (NUMBER_VALUE, JBASE, BASE) FIXED;

/* EACH OF THE FOLLOWING CONTAINS THE INDEX INTO V() OF THE CORRESPONDING
   SYMBOL.   WE ASK:    IF TOKEN = IDENT    ETC.    */
DECLARE (IDENT, STRING, NUMBER, DIVIDE, EOFILE, ORSYMBOL,
   CONCATENATE) FIXED;

/* USED TO SAVE BRANCH ADDRESSES IN DO-LOOP CODE */
DECLARE STEPK FIXED;

/* THE FOLLOWING ARE USED IN THE MACRO EXPANDER.  CONSIDERABLE LOGIC
   IS DEVOTED TO AVOIDING CREATING STRINGS OF LENGTH > 256, THE STRING LIMIT.
*/
DECLARE BALANCE CHARACTER, LB FIXED;
DECLARE MACRO_LIMIT FIXED INITIAL (40), MACRO_NAME(40) CHARACTER,
   MACRO_TEXT(40) CHARACTER, MACRO_INDEX(256) BIT (8),
   TOP_MACRO FIXED INITIAL ("FFFFFFFF");
DECLARE EXPANSION_COUNT FIXED, EXPANSION_LIMIT LITERALLY '300';

/* STOPIT() IS A TABLE OF SYMBOLS WHICH ARE ALLOWED TO TERMINATE THE ERROR
   FLUSH PROCESS.  IN GENERAL THEY ARE SYMBOLS OF SUFFICIENT SYNTACTIC
   HIERARCHY THAT WE EXPECT TO AVOID ATTEMPTING TO START COMPILING AGAIN
   RIGHT INTO ANOTHER ERROR PRODUCING SITUATION.  THE TOKEN STACK IS ALSO
```

```
   FLUSHED DOWN TO SOMETHING ACCEPTABLE TO A STOPIT() SYMBOL.
   FAILSOFT IS A BIT WHICH ALLOWS THE COMPILER ONE ATTEMPT AT A GENTLE
   RECOVERY.   THEN IT TAKES A STRONG HAND.   WHEN THERE IS REAL TROUBLE
   COMPILING IS SET TO FALSE, THEREBY TERMINATING THE COMPILATION.
   MAINLOC IS THE SYMBOL TABLE LOCATION OF COMPACTIFY FOR USE IN ERROR().
*/
DECLARE STOPIT(NT) BIT(1), (FAILSOFT, COMPILING) BIT(1), MAINLOC FIXED;

DECLARE S CHARACTER;  /* A TEMPORARY USED VARIOUS PLACES */

/* THE ENTRIES IN PRMASK() ARE USED TO SELECT OUT PORTIONS OF CODED
   PRODUCTIONS AND THE STACK TOP FOR COMPARISON IN THE ANALYSIS ALGORITHM */
DECLARE PRMASK(5) FIXED INITIAL (0, 0, "FF", "FFFF", "FFFFFF", "FFFFFFFF");

/* SUBSTR(HEXCODES, I, 1) IS THE HEXADECIMAL CODE LETTER FOR I  */
DECLARE HEXCODES CHARACTER INITIAL ('0123456789ABCDEF');

/*THE PROPER SUBSTRING OF POINTER IS USED TO PLACE AN  \  UNDER THE POINT
   OF DETECTION OF AN ERROR DURING COMPILATION.  IT MARKS THE LAST CHARACTER
   SCANNED.  */
DECLARE POINTER CHARACTER INITIAL ('
                                   \');
DECLARE (COUNT$STACK, COUNT$SCAN, COUNT$RR, COUNT$RX, COUNT$FORCE,
   COUNT$ARITH, COUNT$STORE, COUNT$FIXBFW, COUNT$FIXD, COUNT$FIXCHW,
   COUNT$GETD, COUNT$GETC, COUNT$FIND) FIXED;

/* RECORD THE TIMES OF IMPORTANT POINTS DURING COMPILATION */
DECLARE CLOCK(5) FIXED;

/* COUNT THE NUMBER OF COMPARISONS OF IDENTIFIERS IN SYMBOL TABLE LOOK-UPS
   THIS CAN, IN GENERAL, BE EXPECTED TO BE A SUBSTANTIAL PART OF RUN TIME.
*/
DECLARE IDCOMPARES FIXED, STATEMENT_COUNT FIXED;
DECLARE TRUELOC FIXED;  /* ADDRESS OF INTEGER 1 IN DATA AREA */
DECLARE COMPLOC FIXED; /*  THE ADDRESS OF ALL ONES MASK FOR COMPLEMENT */
DECLARE CATCONST FIXED;   /* ADDRESS OF 2**24  */
DECLARE BASEDATA FIXED;   /*  BASE REGISTER INITIALIZATION ADDRESS */

/*  THE EMITTER  ARRAYS  */


/*************************************************************************

   WARNING:  THE EMITTER ARRAYS "CODE", "DATA", AND "STRINGS" ARE
DEPENDENT ON THE HARDWARE DEVICES AVAILABLE FOR SCRATCH STORAGE.  THE
LITERAL CONSTANT "DISKBYTES" SHOULD BE EQUAL TO THE BLOCKSIZE OF THESE FILES
AS ESTABLISHED IN DCB'S IN THE SUBMONITOR.

SUGGESTED VALUES:
        FOR LARGE CORE:                FOR SMALL CORE DISKBYTES = 400.

   2311     DISKBYTES = 3600
   2314     DISKBYTES = 7200
   2321     DISKBYTES = 2000

   THIS VERSION OF XCOM NEEDS THREE SCRATCH FILES:
```

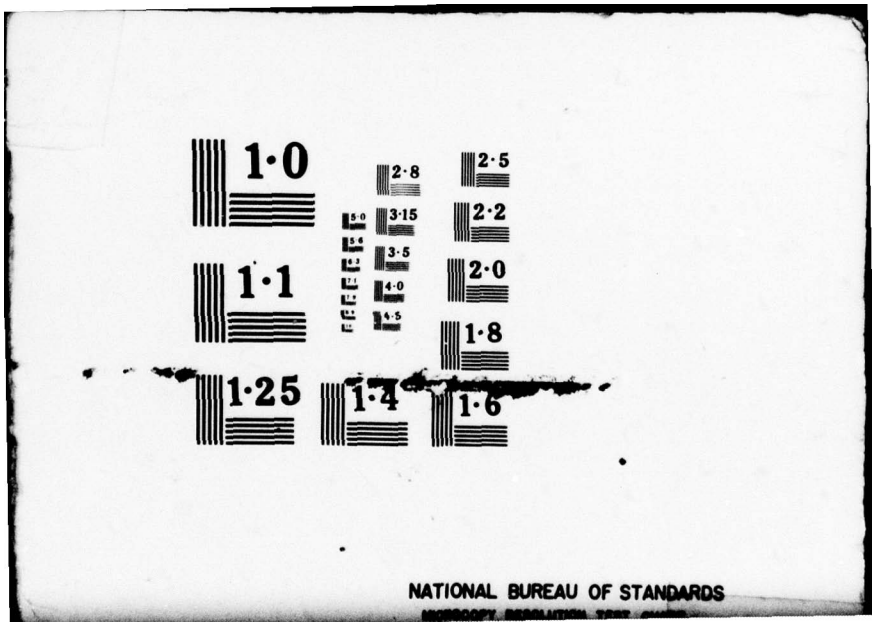
```
      1         COMPILED CODE TEMPORARY
      2         COMPILED DATA TEMPORARY
      3         CHARACTER STRING TEMPORARY
      1         BINARY PROGRAM OUTPUT

 ********************************************************************************/

DECLARE DISKBYTES LITERALLY '3600';    /*  2311  DISKS  */
     /* SIZE OF SCRATCH FILE BLOCKS IN BYTES */
   DECLARE CODEMAX FIXED;        /* FORCES CODE TO WORD BOUNDARY */
   DECLARE CODE (DISKBYTES) BIT(8);
   DECLARE DATAMAX FIXED;     /* FORCES DATA TO WORD BOUNDARY */
   DECLARE DATA (DISKBYTES) BIT(8);
   DECLARE STRNGMX FIXED;             /*  AND FORCE STRINGS TO BE ALIGNED  */
   DECLARE STRINGS (DISKBYTES) BIT(8); /* BUFFER FOR COMPILED STRINGS  */


   /*  CODEMAX  IS THE # OF RECORDS OF CODE GENERATED
       DATAMAX IS THE NUMBER OF RECORDS OF DATA GENERATED
   */

   DECLARE CODEFILE FIXED INITIAL(1);    /* FILE FOR BINARY CODE, AND */
   DECLARE DATAFILE FIXED INITIAL (2);    /* SCRATCH FILE FOR DATA */
   DECLARE STRINGFILE FIXED INITIAL (3);
      /* SCRATCH FILE FOR CHARACTER STRINGS */

   DECLARE (PPORG, PPLIM, DPORG, DPLIM, CURCBLK, CURDBLK, CURSBLK, CHPORG,
      CHPLIM, STRINGMAX, SHORTDFIX, SHORTCFIX, LONGDFIX, LONGCFIX, FCP) FIXED;

   DECLARE LIMITWORD FIXED;
   DECLARE STRING_RECOVER FIXED;


   DECLARE XPLCOMP FIXED; /* RELOCATED ADDR OF THE COMPACTIFY MOD */
   DECLARE XPLMON FIXED;  /* RELOCATED ADDR OF THE XPL SUB MONITOR */
   DECLARE SAVE   FIXED;  /* CONTAINS ADDR OF SAVE AREA FOR EXTERNAL CALLS */
   DECLARE PARMLOC FIXED; /* CONTAINS ADDRESS OF THE PARAMETER LIST PASS
                             FROM AN EXTERNAL ROUTINE TO A PUBLIC PROC */
   DECLARE STRMOVE FIXED;  /* ADDRESS OF COMMON STRING MOVE ROUTINE */
   DECLARE CATENTRY FIXED;    /*  ENTRY TO CATENATE ROUTINE */
   DECLARE STRN FIXED;  /* ADDRESS OF TEMP IN STRING TO NUMBER ROUTINE */
   DECLARE STRL FIXED;  /* ADDRESS OF LAST STRING COMPUTED FOR OPTIMIZING \\ */
   DECLARE FREEPOINT FIXED INITIAL(4); /*OFFSET TO TOP OF STRING AREA */
   DECLARE FREEBASE FIXED INITIAL (8);  /* OFFSET FROM R13 */
   DECLARE FREELIMIT FIXED INITIAL (12); /* OFFSET FROM R13 */
   DECLARE DESCL    FIXED INITIAL (16); /* OFFSET TO # OF DESCRIPTORS */
   DECLARE NMBRNTRY  FIXED;     /*  ENTRY TO BINARY TO CHAR. CONVERSION */
   DECLARE MOVER FIXED;   /*  ADDRESS OF MOVE TEMPLATE  */
   DECLARE BASES (15) FIXED;    /*  THE VALUE OF THE BASE REGISTERS */
   DECLARE AVAIL FIXED INITIAL (2);
   DECLARE DESC(1024) FIXED;  /* STRING DESCRIPTORS, REG 13 RELATIVE */

   /*  360 REGISTER ASSIGNMENTS:
            0     SCRATCH
            1-3   ACCUMULATORS
            4-11  DATA ADDRESSING
```

AD-A062 147 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO F/G 9/2
ENHANCEMENT AND EXTENSIONS TO THE XPL COMPILER GENERATOR SYSTEM--ETC(U)
AUG 78 G R FELDBAUER

UNCLASSIFIED AFIT-CI-79-85 NL

2 OF 2
AD
A062 147

END
DATE
FILMED
3 -79
DDC


1·0
1·1
1·25
2·8
3·15
3·5
4·0
4·5
5·0
1·4
2·5
2·2
2·0
1·8
1·6
NATIONAL BUREAU OF STANDARDS

```
            12    BRANCH REGISTER
            13    STRING DESCRIPTOR AREA BASE
            14    PROGRAM BASE
            15    POINTS TO ENTRY OF I/O PACKAGE
   */

   DECLARE CALLREG LITERALLY '"F"'; /* REGISTER FOR IO ROUTINES OF SUBMONITOR */
   DECLARE BRCHREG LITERALLY '"E"';  /* REGISTER FOR BRANCHING  */
   DECLARE SBR LITERALLY '"D"';  /* STRING BASE REGISTER TO ADDRESS DESCRIPT. */
   DECLARE PBR LITERALLY '"C"';  /* PROGRAM BASE REGISTER POINTS TO CODE  */
   DECLARE DBR LITERALLY '"B"';  /* FIRST DATA BASE REGISTER  */
   DECLARE PROGRAMSIZE LITERALLY '25';  /* NUMBER OF 4096 BYTE PAGES ALLOWED */
   DECLARE LASTBASE FIXED;   /*  KEEP TRACK OF ALLOCATION OF REG 11 - 4 */
   DECLARE TARGET_REGISTER FIXED;
   DECLARE MASKF000 BIT(32);
   DECLARE ADREG FIXED, ADRDISP FIXED;    /* GLOBALS FOR FINDADDRESS */
   DECLARE RTNADR FIXED;  /*  WHERE THE PRESENT RETURN ADDRESS IS STORED */
   DECLARE RETURNED_TYPE BIT (8);
   DECLARE TEMP(3) FIXED ;          /*  STORAGE FOR SAVE_REGISTERS  */
   DECLARE (DP, PP, CHP, DSP, NEWDP, NEWDSP) FIXED; /* EMITTER POINTERS */
   DECLARE ITYPE FIXED;   /*  INITIALIZATION TYPE  */
   DECLARE STILLCOND FIXED;  /*  REMEMBER CONDITION CODE TEST FOR PEEPHOLE */

   /*  COMMON  IBM  360  OP-CODES  */
   DECLARE OPNAMES CHARACTER INITIAL ('    BALRBCTRBCR LPR LNR LTR LCR NR  CLR O
R  XR  LR  CR  AR  SR  MR  DR  ALR SLR LA  STC IC  EX  BAL BCT BC  CVD CVB ST  N
   CL  O   X   L   C   A   S   M   D   AL  SL  SRL SLL SRA SLA SRDLSLDLSRDASLDAS
TM TM  MVI NI  CLI OI  XI  LM  MVC STH LH  ');
   DECLARE OPER(255) BIT(8) INITIAL(
 /*0**/   0,  0,  0,  0,  0,  4,  8, 12,  0,  0,  0,  0,  0,  0,  0,  0,
 /*1**/  16, 20, 24, 28, 32, 36, 40, 44, 48, 52, 56, 60, 64, 68, 72, 76,
 /*2**/   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 /*3**/   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 /*4**/ 236, 80, 84, 88, 92, 96,100,104,240,  0,  0,  0,  0,  0,108,112,
 /*5**/ 116,  0,  0,  0,120,124,128,132,136,140,144,148,152,156,160,164,
 /*6**/   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 /*7**/   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 /*8**/   0,  0,  0,  0,  0,  0,  0,  0,168,172,176,180,184,188,192,196,
 /*9**/ 200,204,208,  0,212,216,220,224,228,  0,  0,  0,  0,  0,  0,  0,
 /*A**/   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 /*B**/   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 /*C**/   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 /*D**/   0,  0,232,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 /*E**/   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
 /*F**/   0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0);
 /*       *0  *1  *2  *3  *4  *5  *6  *7  *8  *9  *A  *B  *C  *D  *E  *F  */
   DECLARE OP_CODE CHARACTER;  /* FOR DEBUG PRINTOUT */
   DECLARE COMMUTATIVE(63) BIT(1);  /* RECORD WHICH OPERATORS ARE COMMUTATIVE */

   /* COMMONLY USED /360 OPERATION CODES */
   DECLARE BC FIXED INITIAL ("47"), BCR FIXED INITIAL ("07");
   DECLARE BAL FIXED INITIAL ("45"), BALR FIXED INITIAL ("05");
   DECLARE LOAD FIXED INITIAL ("58"), STORE FIXED INITIAL ("50");
   DECLARE CMPR FIXED INITIAL ("59"), CMPRR FIXED INITIAL ("19");
   DECLARE LA FIXED INITIAL ("41");
   DECLARE STM  FIXED INITIAL ("90"), LM FIXED INITIAL ("98");
```

```
/* THE FOLLOWING ARE USED TO HOLD ADDRESS PAIRS IN THE EMITTER FOR \\ */
DECLARE (A1, A2, B1, B2, T1, T2) FIXED;

/* COMMONLY USED STRINGS */
DECLARE X1 CHARACTER INITIAL(' '), X4 CHARACTER INITIAL('    ');
DECLARE X0 CHARACTER INITIAL(''); /* NULL CHARACTER STRING CONSTANT */
DECLARE EQUALS CHARACTER INITIAL (' = '), PERIOD CHARACTER INITIAL ('.');
DECLARE COMMA CHARACTER INITIAL (',');
DECLARE LPAREN CHARACTER INITIAL('(');
DECLARE RPAREN CHARACTER INITIAL(')');
DECLARE OR CHARACTER INITIAL('\');
DECLARE CON_E CHARACTER INITIAL('E');
DECLARE HYPHEN CHARACTER INITIAL('''');

/* TEMPORARIES USED THROUGHOUT THE COMPILER */
DECLARE CHAR_TEMP CHARACTER;
DECLARE (I, J, K, L) FIXED;

DECLARE TRUE LITERALLY '1', FALSE LITERALLY '0', FOREVER LITERALLY 'WHILE 1';

/*  SYMBOL  TABLE  VARIABLES  */

DECLARE HALFWORD      LITERALLY  '1',
        LABELTYPE     LITERALLY  '2',
        ACCUMULATOR   LITERALLY  '3',
        VARIABLE      LITERALLY  '4',
        CONSTANT      LITERALLY  '5',
        CONDITION     LITERALLY  '6',
        CHRTYPE       LITERALLY  '7',
        FIXEDTYPE     LITERALLY  '8',
        BYTETYPE      LITERALLY  '9',
        FORWARDTYPE   LITERALLY '10',
        DESCRIPT      LITERALLY '11',
        SPECIAL       LITERALLY '12',
        FORWARDCALL   LITERALLY '13' ,
        CHAR_PROC_TYPE LITERALLY '14',
        EXT_PARM      LITERALLY '15'
        ;
DECLARE TYPENAME(14) CHARACTER INITIAL ('', 'BIT(16)  ', 'LABEL    ', '', '',
   '', '', 'CHARACTER', 'FIXED    ', 'BIT(8)   ', '', '', '', '',
   'CHARACTER PROCEDURE');
DECLARE PROCMARK FIXED;  /* START OF LOCAL VARIABLES IN SYMBOL TABLE */
DECLARE PARCT FIXED;  /* NUMBER OF PARAMETERS TO CURRENT PROCEDURE */
DECLARE NDECSY FIXED;      /* CURRENT NUMBER OF DECLARED SYMBOLS */
/* MAXNDECSY IS THE MAXIMUM OF NDECSY OVER A COMPILATION.  IF MAXNDECSY
   BEGINS TO APPROACH SYTSIZE THEN SYTSIZE SHOULD BE INCREASED */
DECLARE MAXNDECSY FIXED;

DECLARE SYTSIZE LITERALLY '450';  /*   SYMBOL TABLE SIZE   */

/*  THE SYMBOL TABLE IS INITIALIZED WITH THE NAMES OF ALL
    BUILTIN FUNCTIONS AND PSEUDO VARIABLES.  THE PROCEDURE
    INITIALIZE DEPENDS ON THE ORDER AND PLACEMENT OF THESE
    NAMES.  DUE CAUTION SHOULD BE OBSERVED WHILE MAKING CHANGES .
*/
```

```
DECLARE SYT (SYTSIZE) CHARACTER          /*  VARIABLE NAME */
   INITIAL ('','','MONITOR_LINK','TIME_OF_GENERATION',
      'DATE_OF_GENERATION','COREWORD','COREBYTE','',
      'DESCRIPTOR','NDESCRIPT', 'LENGTH','SUBSTR','BYTE','SHL',
      'SHR','INPUT','OUTPUT','FILE','INLINE','TRACE','UNTRACE',
      'EXIT','TIME','DATE','CLOCK_TRAP','INTERRUPT_TRAP',
      'MONITOR','ADDR','XPLPACK','XPLMON','XPLINIT');
DECLARE SYTYPE (SYTSIZE) BIT (8)   /* TYPE OF THE VARIABLE */
   INITIAL (0,0,FIXEDTYPE,FIXEDTYPE,FIXEDTYPE,FIXEDTYPE,
      BYTETYPE,FIXEDTYPE,FIXEDTYPE,FIXEDTYPE,SPECIAL,SPECIAL,
      SPECIAL,SPECIAL,SPECIAL,SPECIAL,SPECIAL,SPECIAL,SPECIAL,
      SPECIAL,SPECIAL,SPECIAL,SPECIAL,SPECIAL,SPECIAL,SPECIAL,
      SPECIAL,SPECIAL,SPECIAL,SPECIAL,SPECIAL);
DECLARE SYBASE (SYTSIZE) BIT (4) INITIAL(0,0,DBR,DBR,DBR,0,0,
   DBR,4,DBR,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,DBR,0,0);
DECLARE SYDISP (SYTSIZE) BIT (12)  /* DISPLACEMENT FOR VARIABLE */
   INITIAL (0,0,0,0,0,0,0,0,0,0,1,2,3,4,5,6,7,8,  10,11,12,13,14,
      15,16,17,18,19,0,0);
DECLARE SYTCO(SYTSIZE) BIT (16);     /* COUNT OF REFERENCES TO SYMBOLS */
DECLARE SYESDTYPE(SYTSIZE) BIT(8); /* USED FOR ESD CREATION */
DECLARE SYRLDADDR(SYTSIZE) FIXED ; /* ADDR TO BE RELOCATED  */
DECLARE SYLEN(SYTSIZE)     FIXED ; /* LENGTH OF LABELED COMMON VARIABLE */
DECLARE DECLARED_ON_LINE(SYTSIZE) BIT(16);

/*  THE COMPILER STACKS DECLARED BELOW ARE USED TO DRIVE THE SYNTACTIC
   ANALYSIS ALGORITHM AND STORE INFORMATION RELEVANT TO THE INTERPRETATION
   OF THE TEXT.  THE STACKS ARE ALL POINTED TO BY THE STACK POINTER SP.  */

DECLARE STACKSIZE LITERALLY '35';  /* SIZE OF STACK  */
DECLARE PARSE_STACK (STACKSIZE) BIT(8); /* TOKENS OF THE PARTIALLY PARSED
                                           TEXT */
DECLARE TYPE (STACKSIZE) BIT(8);  /* OPERAND TYPE FOR EXPRESSIONS */
DECLARE REG (STACKSIZE) BIT(8);   /* ASSOCIATED GENERAL REGISTER */
DECLARE INX (STACKSIZE) BIT(8);   /* ASSOCIATED INDEX REGISTER */
DECLARE CNT (STACKSIZE) BIT(8);   /* ANNY COUNT, PARAMETERS, SUBSCRIPTS ...*/
DECLARE VAR (STACKSIZE) CHARACTER;/* EBCDIC NAME OF ITEM */
DECLARE FIXL (STACKSIZE) FIXED;   /* FIXUP LOCATION */
DECLARE FIXV (STACKSIZE) FIXED;   /* FIXUP VALUE */
DECLARE PPSAVE (STACKSIZE) FIXED; /* ASSOCIATED PROGRAM POINTER */

/* SP POINTS TO THE RIGHT END OF THE REDUCIBLE STRING IN THE PARSE STACK,
   MP POINTS TO THE LEFT END, AND
   MPP1 = MP+1.
*/
DECLARE (SP, MP, MPP1) FIXED;

/* DECLARE STATEMENTS AND CASE STATEMENTS REQUIRE AN AUXILIARY STACK */
DECLARE CASEP FIXED;   /* POINTS TO THE CURRENT POSITION IN CASESTACK */
DECLARE CASELIMIT LITERALLY '255', CASESTACK(CASELIMIT) FIXED;
DECLARE INSTRUCT (255) BIT(16);  /*  INSTRUCTION USE COUNTERS */

   /* ARRAYS TO HOLD FIXUPS DURING COMPILATION */

/* FCLIM IS THE NUMBER OF FIXUPS THAT CAN BE RECORDED BEFORE THEY ARE MADE */
DECLARE FCLIM LITERALLY '100';
```

```
   DECLARE FIXCADR (FCLIM) FIXED;      /* ADDRESS OF CODE FIXUP  */
   DECLARE FIXCB1 (FCLIM) BIT(8);      /* 1ST BYTE OF CODE FIXUP  */
   DECLARE FIXCB2 (FCLIM) BIT(8);      /* 2ND BYTE OF CODE FIXUP  */
   DECLARE DCLRM CHARACTER INITIAL ('IDENTIFIER LIST TOO LONG');



   DECLARE H00 BIT(8) INITIAL (0);
   DECLARE H02 BIT(8) INITIAL (2);
   DECLARE H40 BIT(8) INITIAL ("40");
   DECLARE PROC_NAME CHARACTER  INITIAL ('MAIN');
   DECLARE MAIN_PTR FIXED;  /* SYMBOL TABLE POINTER TO THE FIRST PUBLIC PROC*/
   DECLARE CODE_ESDID FIXED;
   DECLARE ESDID FIXED;
   DECLARE ESDTYPE BIT(8);   /* IDENTIFIES THE TYPE OF ESD ENTRY */
   DECLARE ESD_FIRST BIT (16) INITIAL ("4040");
   DECLARE WORKBYTE BIT (8);
   DECLARE WORKWORD FIXED;
   DECLARE STRREG FIXED;  /* TEMP SAVE AREA FOR STRING ADDRESS REG */
   DECLARE MOVEFLAG BIT(1) INITIAL (TRUE); /* USED IN MOVING COMMON STRINGS*/


   /*              P R O C E D U R E S                              */


PAD:
   PROCEDURE (STRING, WIDTH) CHARACTER;
      DECLARE STRING CHARACTER, (WIDTH, L) FIXED;

      L = LENGTH(STRING);
      IF L >= WIDTH THEN RETURN STRING;
      ELSE RETURN STRING \\ SUBSTR(X70, 0, WIDTH-L);
   END PAD;

I_FORMAT:
   PROCEDURE (NUMBER, WIDTH) CHARACTER;
      DECLARE (NUMBER, WIDTH, L) FIXED, STRING CHARACTER;

      STRING = NUMBER;
      L = LENGTH(STRING);
      IF L >= WIDTH THEN RETURN STRING;
      ELSE RETURN SUBSTR(X70, 0, WIDTH-L) \\ STRING;
   END I_FORMAT;

ERROR:
   PROCEDURE(MSG, SEVERITY);
      /* PRINTS AND ACCOUNTS FOR ALL ERROR MESSAGES */
      /* IF SEVERITY IS NOT SUPPLIED, 0 IS ASSUMED */
      DECLARE MSG CHARACTER, SEVERITY FIXED;
      ERROR_COUNT = ERROR_COUNT + 1;
      /* IF LISTING IS SUPPRESSED, FORCE PRINTING OF THIS LINE */
      IF ^ CONTROL(BYTE('L')) THEN
         OUTPUT = I_FORMAT (CARD_COUNT, 5) \\ OR \\ BUFFER \\ OR;
      OUTPUT = SUBSTR(POINTER, TEXT_LIMIT+LB-CP+MARGIN_CHOP);
      /* SEVERITY(-1) IS A PORNOGRAPHIC WAY OF OBTAINING THE RETURN ADDRESS */
```

```
      OUTPUT = '*** ERROR, ' \\ MSG \\
            '  LAST PREVIOUS ERROR WAS DETECTED ON LINE ' \\
            PREVIOUS_ERROR ;
      PREVIOUS_ERROR = CARD_COUNT;
      IF SEVERITY > 0 THEN
         IF SEVERE_ERRORS > 25 THEN
            DO;
               OUTPUT = '*** TOO MANY SEVERE ERRORS, COMPILATION ABORTED ***';
               COMPILING = FALSE;
            END;
         ELSE SEVERE_ERRORS = SEVERE_ERRORS + 1;
   END ERROR;


   /*                        FILE HANDLING PROCEDURES                          */


GETDATA:
   PROCEDURE;
      /* HANDLE SCRATCH STORAGE ALLOCATION FOR THE DATA ARRAY  */
      DECLARE I FIXED;
      COUNT$GETD = COUNT$GETD + 1;
      FILE(DATAFILE,CURDBLK) = DATA;      /*  WRITE OUT CURRENT BLOCK */
      CURDBLK = DP / DISKBYTES;           /* CALCULATE NEW BLOCK NUMBER */
      DPORG = CURDBLK * DISKBYTES;
      DPLIM = DPORG + DISKBYTES;
      IF CURDBLK <= DATAMAX THEN
            DATA = FILE(DATAFILE,CURDBLK);
      ELSE
         DO;
            /*  ZERO OUT THE NEW DATA BLOCK  */
            DO I = 1 TO SHR(DISKBYTES,2);
               DATAMAX(I) = 0;
            END;
            DO DATAMAX = DATAMAX + 1 TO CURDBLK - 1;
               FILE(DATAFILE,DATAMAX)= DATA;
            END;
         END;
   END  GETDATA;

GETCODE:
   PROCEDURE;
      /*  HANDLE SCRATCH STORAGE ALLOCATION FOR THE CODE ARRAY */
      DECLARE I FIXED;
      COUNT$GETC = COUNT$GETC + 1;
      FILE(CODEFILE,CURCBLK) = CODE;
      CURCBLK = PP / DISKBYTES;        /* CALCULATE NEW BLOCK NUMBER */
      PPORG = CURCBLK * DISKBYTES;
      PPLIM = PPORG + DISKBYTES;
      IF CURCBLK <= CODEMAX THEN
               CODE = FILE(CODEFILE,CURCBLK);
      ELSE
         DO;
         /*  ZERO OUT THE NEW CODE BLOCK */
            DO I = 1 TO SHR(DISKBYTES,2);
```

```
            CODEMAX(I) = 0;
         END;
         DO CODEMAX = CODEMAX + 1 TO CURCBLK - 1;
            FILE(CODEFILE,CODEMAX) = CODE;
         END;
      END;
   END  GETCODE;

GETSTRINGS:
   PROCEDURE;
      /* HANDLE SCRATCH STORAGE ALLOCATION FOR STRING ARRAY */
      FILE(STRINGFILE, CURSBLK) = STRINGS;     /* WRITE INTO THE FILE */
      CURSBLK = CHP / DISKBYTES;               /* COMPUTE NEW BLOCK NUMBER */
      CHPORG = CURSBLK*DISKBYTES;              /* NEW BLOCK ORIGIN */
      CHPLIM = CHPORG + DISKBYTES;             /* NEW UPPER BOUND */
      IF CURSBLK <= STRINGMAX THEN
         STRINGS = FILE(STRINGFILE,CURSBLK);   /* READ BACK FROM FILE */
      ELSE
         DO STRINGMAX = STRINGMAX+1 TO CURSBLK - 1;
            FILE(STRINGFILE,STRINGMAX) = STRINGS;
            /* FILL OUT FILE SO NO GAPS EXIST */
         END;
   END GETSTRINGS;


  /*                    CARD IMAGE HANDLING PROCEDURE                    */


GET_CARD:
   PROCEDURE;
      /* DOES ALL CARD READING AND LISTING                                */
      DECLARE I FIXED, (TEMP, TEMPO, REST) CHARACTER, READING BIT(1);
      IF LB > 0 THEN
         DO;
            TEXT = BALANCE;
            TEXT_LIMIT = LB - 1;
            CP = 0;
            RETURN;
         END;
      EXPANSION_COUNT = 0;    /* CHECKED IN SCANNER MACRO EXPANSION */
      BUFFER = INPUT;
      IF LENGTH(BUFFER) = 0 THEN
         DO; /* SIGNAL FOR EOF */
            CALL ERROR ('EOF MISSING OR COMMENT STARTING IN COLUMN 1.',1);
            BUFFER = PAD (' /*''/* */ EOF;END;EOF', 80);
         END;
      ELSE CARD_COUNT = CARD_COUNT + 1;  /* USED TO PRINT ON LISTING */
      IF MARGIN_CHOP > 0 THEN
         DO; /* THE MARGIN CONTROL FROM DOLLAR \ */
            I = LENGTH(BUFFER) - MARGIN_CHOP;
            REST = SUBSTR(BUFFER, I);
            BUFFER = SUBSTR(BUFFER, 0, I);
         END;
      ELSE REST = X0;
      TEXT = BUFFER;
```

```
      TEXT_LIMIT = LENGTH(TEXT) - 1;
      IF CONTROL(BYTE('M')) THEN
         OUTPUT = I_FORMAT(PP,7) \\ SUBSTR(X70,1,20) \\ BUFFER;
      ELSE IF CONTROL(BYTE('L')) THEN
         DO;
            REST = I_FORMAT (PP, 6) \\ REST;
            OUTPUT = I_FORMAT (CARD_COUNT, 5) \\ OR \\ BUFFER \\ OR \\
               REST \\ CURRENT_PROCEDURE \\ INFORMATION;
         END;
      INFORMATION = X0;
      CP = 0;
   END GET_CARD;


   /*                THE SCANNER PROCEDURES                */


CHAR:
   PROCEDURE;
      /* USED FOR STRINGS TO AVOID CARD BOUNDARY PROBLEMS */
      CP = CP + 1;
      IF CP <= TEXT_LIMIT THEN RETURN;
      CALL GET_CARD;
   END CHAR;

DEBLANK:
   PROCEDURE;
      /* USED BY BCHAR */
      CALL CHAR;
      DO WHILE BYTE(TEXT, CP) = BYTE(X1);
         CALL CHAR;
      END;
   END DEBLANK;

BCHAR:
   PROCEDURE;
      /* USED FOR BIT STRINGS */
      DO FOREVER;
         CALL DEBLANK;
         CH = BYTE(TEXT, CP);
         IF CH ^= BYTE(LPAREN) THEN RETURN;
         /*  (BASE WIDTH)  */
         CALL DEBLANK;
         JBASE = BYTE(TEXT, CP) - "F0";  /* WIDTH */
         IF JBASE < 1 \ JBASE > 4 THEN
            DO;
               CALL ERROR ('ILLEGAL BIT STRING WIDTH: ' \\ SUBSTR(TEXT, CP, 1));
               JBASE = 4;  /* DEFAULT WIDTH FOR ERROR */
            END;
         BASE = SHL(1, JBASE);
         CALL DEBLANK;
         IF BYTE(TEXT, CP) ^= BYTE(RPAREN) THEN
            CALL ERROR ('MISSING ) IN BIT STRING', 0 );
      END;
   END BCHAR;
```

```
BUILD_BCD:
   PROCEDURE (C);
      DECLARE C BIT(8);
      IF LENGTH(BCD) > 0 THEN
         BCD = BCD \\ X1;
      ELSE
         BCD = SUBSTR(X1 \\ X1, 1);
      /* FORCE BCD TO THE TOP OF FREE STRING AREA AND INCREASE LENGTH
          BY ONE  */
      /* THIS LINE DEPENDS UPON THE IMPLEMENTATION OF XPL STRINGS */
      COREBYTE(FREEPOINT-1) = C;
   END BUILD_BCD;

SCAN:
   PROCEDURE;
      DECLARE (S1, S2) FIXED;
      DECLARE LSTRNGM CHARACTER INITIAL('STRING TOO LONG');
      COUNT#SCAN = COUNT#SCAN + 1;
      FAILSOFT = TRUE;
      BCD = X0;  NUMBER_VALUE = 0;
   SCAN1:
      DO FOREVER;
         IF CP > TEXT_LIMIT THEN CALL GET_CARD;
         ELSE
            DO; /* DISCARD LAST SCANNED VALUE */
               TEXT_LIMIT = TEXT_LIMIT - CP;
               TEXT = SUBSTR(TEXT, CP);
               CP = 0;
            END;
         /*  BRANCH ON NEXT CHARACTER IN TEXT                    */
         DO CASE CHARTYPE(BYTE(TEXT));

            /*  CASE 0  */

            /* ILLEGAL CHARACTERS FALL HERE  */
            CALL ERROR ('ILLEGAL CHARACTER: ' \\ SUBSTR(TEXT, 0, 1));

            /*  CASE 1  */

            /*  BLANK  */
            DO;
               CP = 1;
               DO WHILE BYTE(TEXT, CP) = BYTE(X1) & CP <= TEXT_LIMIT;
                  CP = CP + 1;
               END;
               CP = CP - 1;
            END;

            /*  CASE 2  */

            /*  STRING QUOTE ('):  CHARACTER STRING  */
            DO FOREVER;
               TOKEN = STRING;
               S1 = 1;
               CALL CHAR;
               DO WHILE BYTE(TEXT, CP) ^= BYTE(HYPHEN);
```

```
      IF CP <= TEXT_LIMIT THEN
         CP = CP+1;
      ELSE
         DO; /* STRING BROKEN ACROSS CARD BOUNDARY */
            IF LENGTH(BCD) + CP > 257 THEN
               DO;
                  CALL ERROR(LSTRNGM  ,0);
                  RETURN;
               END;
            IF CP > S1 THEN
               BCD = BCD \\ SUBSTR(TEXT,S1,CP-S1);
            TEXT = X1;
            CP = 0;
            CALL GET_CARD;
   S1 = 0;
         END;
   END;
   IF LENGTH(BCD) + CP > 257 THEN
      DO;
         CALL ERROR(LSTRNGM,0);
         RETURN;
      END;
   IF CP > S1 THEN
      BCD = BCD \\ SUBSTR(TEXT,S1,CP-S1);
   CALL CHAR;
   IF BYTE(TEXT, CP) ^= BYTE(HYPHEN) THEN
      RETURN;
   IF LENGTH(BCD) > 255 THEN
      DO;
         CALL ERROR(LSTRNGM,0);
         RETURN;
      END;
   BCD = BCD \\ HYPHEN;
   TEXT_LIMIT = TEXT_LIMIT - CP;
   TEXT = SUBSTR(TEXT,CP);
   CP = 0;    /* PREPARE TO RESUME SCANNING STRING */
END;

/*  CASE 3  */

DO;       /*  BIT QUOTE('):  BIT STRING  */
   JBASE = 4;  BASE = SHL(1, JBASE);  /* DEFAULT WIDTH  */
   TOKEN = NUMBER;  /* ASSUME SHORT BIT STRING */
   S1 = 0;
   CALL BCHAR;
   DO WHILE CH ^= BYTE('''');
      S1 = S1 + JBASE;
      IF CH >= "F0" THEN S2 = CH - "F0";  /* DIGITS */
      ELSE S2 = CH - "B7";                /* LETTERS */
      IF S2 >= BASE \ S2 < 0 THEN
         CALL ERROR ('ILLEGAL CHARACTER IN BIT STRING: '
         \\ SUBSTR(TEXT, CP, 1));
      IF S1 > 32 THEN TOKEN = STRING;     /* LONG BIT STRING */
      IF TOKEN = STRING THEN
         DO WHILE S1 - JBASE >= 8;
            IF LENGTH(BCD) > "FF" THEN
```

```
            DO;
               CALL ERROR (LSTRNGM, 0);
               RETURN;
            END;
         S1 = S1 - 8;
         CALL BUILD_BCD (SHR(NUMBER_VALUE, S1-JBASE));
      END;
   NUMBER_VALUE = SHL(NUMBER_VALUE, JBASE) + S2;
   CALL BCHAR;
END;     /* OF DO WHILE CH...  */
CP = CP + 1;
IF TOKEN = STRING THEN
   IF LENGTH(BCD) > "FF" THEN CALL ERROR (LSTRNGM, 0);
   ELSE CALL BUILD_BCD (SHL(NUMBER_VALUE, 8 - S1));
RETURN;
END;

/*  CASE 4  */

DO FOREVER;  /* A LETTER:  IDENTIFIERS AND RESERVED WORDS */
   DO CP = CP + 1 TO TEXT_LIMIT;
      IF NOT_LETTER_OR_DIGIT(BYTE(TEXT, CP)) THEN
         DO;  /* END OF IDENTIFIER  */
            IF CP > 0 THEN BCD = BCD \\ SUBSTR(TEXT, 0, CP);
            S1 = LENGTH(BCD);
            IF S1 > 1 THEN IF S1 <= RESERVED_LIMIT THEN
               /* CHECK FOR RESERVED WORDS */
               DO I = V_INDEX(S1-1) TO V_INDEX(S1) - 1;
                  IF BCD = V(I) THEN
                     DO;
                        TOKEN = I;
                        RETURN;
                     END;
               END;
            DO I = MACRO_INDEX(S1-1) TO MACRO_INDEX(S1) - 1;
               IF BCD = MACRO_NAME(I) THEN
                  DO;
                     BCD = MACRO_TEXT(I);
                     IF EXPANSION_COUNT < EXPANSION_LIMIT THEN
                        EXPANSION_COUNT = EXPANSION_COUNT + 1;
                     ELSE OUTPUT =
                        '*** WARNING, TOO MANY EXPANSIONS FOR ' \\
                        MACRO_NAME(I) \\ ' LITERALLY: ' \\ BCD;
                     TEXT = SUBSTR(TEXT, CP);
                     TEXT_LIMIT = TEXT_LIMIT - CP;
                     IF LENGTH(BCD) + TEXT_LIMIT > 255 THEN
                        DO;
                           IF LB + TEXT_LIMIT > 255 THEN
                             CALL ERROR('MACRO EXPANSION TOO LONG');
                           ELSE
                              DO;
                                 BALANCE = TEXT \\ BALANCE;
                                 LB = LENGTH(BALANCE);
                                 TEXT = BCD;
                              END;
                        END;
```

```
                        ELSE TEXT = BCD \\ TEXT;
                        BCD = X0;
                        TEXT_LIMIT = LENGTH(TEXT) - 1;
                        CP = 0;
                        GO TO SCAN1;
                     END;
               END;
               /*  RESERVED WORDS EXIT HIGHER: THEREFORE <IDENTIFIER>*/
               TOKEN = IDENT;
               RETURN;
            END;
      END;
      /*  END OF CARD  */
      BCD = BCD \\ TEXT;
      CALL GET_CARD;
      CP = -1;
   END;

   /*  CASE 5  */

   DO;       /*  DIGIT:  A NUMBER  */
      TOKEN = NUMBER;
      DO FOREVER;
         DO CP = CP TO TEXT_LIMIT;
            S1 = BYTE(TEXT, CP);
            IF S1 < "F0" THEN RETURN;
            NUMBER_VALUE = 10*NUMBER_VALUE + S1 - "F0";
         END;
         CALL GET_CARD;
      END;
   END;

   /*  CASE 6  */

   DO;       /*  A /:  MAY BE DIVIDE OR START OF COMMENT  */
      CALL CHAR;
      IF BYTE(TEXT, CP) ^= BYTE('*') THEN
         DO;
            TOKEN = DIVIDE;
            RETURN;
         END;
      /* WE HAVE A COMMENT  */
      S1, S2 = BYTE(X1);
      DO WHILE S1 ^= BYTE('*') \ S2 ^= BYTE('/');
         IF S1 = BYTE('$') THEN
            DO;  /* A CONTROL CHARACTER  */
               CONTROL(S2) = ^ CONTROL(S2);
               IF S2 = BYTE('T') THEN CALL TRACE;
               ELSE IF S2 = BYTE('U') THEN CALL UNTRACE;
               ELSE IF S2 = BYTE(OR) THEN
                  IF CONTROL(S2) THEN
                     MARGIN_CHOP = TEXT_LIMIT - CP + 1;
                  ELSE
                     MARGIN_CHOP = 0;
            END;
         S1 = S2;
```

```
            CALL CHAR;
            S2 = BYTE(TEXT, CP);
         END;
      END;

      /*  CASE 7  */
      DO;       /*  SPECIAL CHARACTERS  */
         TOKEN = TX(BYTE(TEXT));
         CP = 1;
         RETURN;
      END;

      /*  CASE 8  */
      DO;  /* A \:  MAY BE "OR" OR "CAT"  */
         CALL CHAR;
         IF BYTE(TEXT, CP) = BYTE(OR) THEN
            DO;
               CALL CHAR;
               TOKEN = CONCATENATE;
            END;
         ELSE TOKEN = ORSYMBOL;
         RETURN;
      END;

   END;      /* OF CASE ON CHARTYPE  */
   CP = CP + 1;  /* ADVANCE SCANNER AND RESUME SEARCH FOR TOKEN  */
END;
END SCAN;


/*              ADDRESS AND REGISTER COMPUTATIONS                    */


CHECKBASES:
   PROCEDURE;
      IF ^ COMPILING THEN RETURN;
      IF DP >= BASES(LASTBASE) + 4096 THEN
         DO;
            LASTBASE = LASTBASE - 1;  /* USE REG 11 DOWN TO REG 4 */
            BASES(LASTBASE) = DP & "FFFFFC";
            INFORMATION = INFORMATION \\ ' R' \\ LASTBASE \\ EQUALS \\
               BASES(LASTBASE) \\ PERIOD;
            IF LASTBASE = 3 THEN CALL ERROR('EXCEEDED DATA AREA',1);
         END;
   END  CHECKBASES;

CLEARREGS:
   PROCEDURE;
      /* FREE ALL THE ARITHMETIC REGISTERS  */
      DO I = 0 TO 3;  BASES(I) = AVAIL;  END;
      TARGET_REGISTER = -1;
   END  CLEARREGS;

FINDAC:
   PROCEDURE FIXED;
      /*  FIND AN ACCUMULATOR FOR 32 BIT QUANTITY  */
```

```
      DECLARE I FIXED;
      IF TARGET_REGISTER > -1 THEN IF BASES(TARGET_REGISTER) = AVAIL THEN
         DO;
            BASES(TARGET_REGISTER) = ACCUMULATOR;
            RETURN TARGET_REGISTER;
         END;
      DO I = 1 TO 3;
         IF BASES(I) = AVAIL THEN
            DO;
               BASES(I) = ACCUMULATOR;
               RETURN I;
            END;
      END;
      CALL ERROR('USED ALL ACCUMULATORS',0);
      RETURN 0;
   END  FINDAC;


FINDADDRESS:
   PROCEDURE (ADR);
      /* FIND THE APPROPRIATE BASE AND DISPLACEMENT FOR THE ADDRESS  */
      DECLARE (ADR, I) FIXED;
      COUNT#FIND = COUNT#FIND + 1;
      IF ADR < 0 THEN
         DO;
            ADRDISP = - ADR;
            ADREG = SBR;
            RETURN;
         END;
      IF ADR = 0 THEN
         DO;
            ADREG,ADRDISP = 0;
            RETURN;
         END;
      DO I = LASTBASE TO DBR;
         IF BASES(I) <= ADR & BASES(I)+4096 > ADR THEN
            DO;
               ADRDISP = ADR - BASES(I);
               ADREG = I;
               RETURN;
            END;
      END;
      CALL ERROR('FIND ADDRESS FAILED',1);
      ADREG,ADRDISP = 0;
   END  FINDADDRESS;



  /*                    CODE EMISSION PROCEDURES                    */


EMITCHAR:
   PROCEDURE (C);
      DECLARE C BIT (8);
      /*  SEND ONE 8-BIT CHARACTER TO THE STRING AREA  */
```

```
      IF CONTROL(BYTE(CON_E)) THEN
         OUTPUT = X4 \\ CHP \\ ': CHARACTER = ' \\
            SUBSTR(HEXCODES, SHR(C,4), 1) \\ SUBSTR(HEXCODES, C & "F", 1);
      IF CHP < CHPORG \ CHP >= CHPLIM THEN CALL GETSTRINGS;
      STRINGS(CHP-CHPORG) = C;
      CHP = CHP + 1;
   END  EMITCHAR;

EMITBYTE:
   PROCEDURE (B);
      DECLARE B FIXED;
      /*  EMIT ONE BYTE OF DATA  */
      IF DP < DPORG \ DP >= DPLIM THEN CALL GETDATA;
      DATA(DP-DPORG) = B;
      IF CONTROL(BYTE(CON_E)) THEN
         OUTPUT = X4 \\ DP \\ ': DATA = ' \\
            SUBSTR(HEXCODES, SHR(B,4), 1) \\ SUBSTR(HEXCODES, B & "F", 1);
      DP = DP + 1;
      CALL CHECKBASES;
   END EMITBYTE;

EMITCODEBYTES:
   PROCEDURE (B1,B2);
      DECLARE (B1, B2) BIT(8), I FIXED;
      /*  EMIT TWO BYTES OF CODE  */
      STILLCOND = 0;
      IF PP < PPORG \ PP >= PPLIM THEN CALL GETCODE;
      I = PP - PPORG;
      CODE(I) = B1;              /*  FIRST  BYTE  */
      CODE(I+1) = B2;            /*  SECOND  BYTE  */
      IF CONTROL(BYTE('B')) THEN
         OUTPUT = X4 \\ PP \\ ': CODE = ' \\
            SUBSTR(HEXCODES, SHR(B1,4), 1) \\ SUBSTR(HEXCODES, B1 & "F", 1)
            \\ SUBSTR(HEXCODES, SHR(B2,4), 1) \\ SUBSTR(HEXCODES, B2 & "F",1);
      PP =  PP + 2;
   END  EMITCODEBYTES ;

EMITDATAWORD:
   PROCEDURE(W);
      DECLARE (W, I) FIXED;
      /*  SEND A 32-BIT WORD TO THE DATA ARRAY  */
      DP = (DP + 3) & "FFFFFC";
      IF DP < DPORG \ DP >= DPLIM THEN CALL GETDATA;
      CALL CHECKBASES;
      IF CONTROL(BYTE(CON_E)) THEN
         OUTPUT = X4 \\ DP \\ ': DATA = ' \\ W;
      I = DP - DPORG;
      DATA(I) = SHR(W,24);
      DATA(I+1) = SHR(W,16);
      DATA(I+2) = SHR(W,8);
      DATA(I+3) = W;
      DP = DP + 4;
      CALL CHECKBASES;
   END EMITDATAWORD;

EMITDESC:
```

```
   PROCEDURE (D);
      DECLARE D FIXED;
      /*  SEND 32-BIT DESCRIPTOR TO STRING DESCRIPTOR AREA  */
      IF DSP >= 4096 THEN
         DO;
            CALL ERROR ('TOO MANY STRINGS', 1);
            DSP = 0;
         END;
      IF CONTROL(BYTE(CON_E)) THEN
         OUTPUT = X4 \\ DSP \\ ': DESC = ' \\ SHR(D,24) \\ COMMA \\
            (D & "FFFFFF");
      DESC(SHR(DSP,2)) = D;
      DSP = DSP + 4;
   END EMITDESC;

EMITCONSTANT:
   PROCEDURE(C);
      /* SEE IF C HAS ALREADY BEEN EMITED, AND IF NOT EMIT.  SET UP ADDRESS */
      DECLARE CTAB(100) FIXED, CADD (100) BIT(16), (C, NC, I) FIXED;
      DO I = 1 TO NC;
         IF CTAB(I) = C THEN
            DO;
               ADREG = SHR(CADD(I),12);
               ADRDISP = CADD(I) & "FFF";
               RETURN;
            END;
      END;
      CALL EMITDATAWORD (C);
      CTAB(I) = C;
      CALL FINDADDRESS(DP-4);
      CADD(I) = SHL(ADREG,12) + ADRDISP;
      IF I < 100 THEN NC = I;
      INFORMATION = INFORMATION \\ ' C' \\ I \\ EQUALS \\ C \\ PERIOD;
   END EMITCONSTANT;

EMITRR:
   PROCEDURE (OP, R1, R2);
      DECLARE (OP, R1, R2) FIXED;
      /* EMIT A 16-BIT RR FORMAT INSTRUCTION  */
      COUNT$RR = COUNT$RR + 1;
      IF CONTROL(BYTE(CON_E)) THEN
         DO;
            OP_CODE = SUBSTR(OPNAMES, OPER(OP), 4);
            OUTPUT = X4 \\ PP \\ ': CODE = ' \\ OP_CODE \\ X1 \\ R1
               \\ COMMA \\ R2;
         END;
      CALL EMITCODEBYTES(OP, SHL(R1,4)+R2);
      INSTRUCT(OP) = INSTRUCT(OP) + 1;
   END EMITRR;

EMITRX:
   PROCEDURE (OP, R1, R2, R3, DISP);
      DECLARE (OP, R1, R2, R3, DISP) FIXED;
      /*  EMIT A 32-BIT RX FORMAT INSTRUCTION */
      COUNT$RX = COUNT$RX + 1;
      IF CONTROL(BYTE(CON_E)) THEN
```

```
      DO;
         OP_CODE = SUBSTR(OPNAMES, OPER(OP), 4);
         OUTPUT = X4 \\ PP \\ ': CODE = ' \\ OP_CODE \\ X1 \\ R1
            \\ COMMA \\ DISP \\ LPAREN \\ R2 \\ COMMA \\ R3 \\ RPAREN;
      END;
   CALL EMITCODEBYTES(OP, SHL(R1,4)+R2);
   CALL EMITCODEBYTES(SHL(R3,4)+SHR(DISP,8), DISP & "FF");
   INSTRUCT(OP) = INSTRUCT(OP) + 1;
END EMITRX;

/*                          FIXUP PROCEDURES                          */


INSERT_CODE_FIXUPS:
   PROCEDURE;
      /* EMPTY THE FIXUP TABLE, EITHER FOR LOADING OR BECAUSE OF
         TABLE OVERFLOW */
      DECLARE (I, J, L, FXLIM, T1, K) FIXED;
      DECLARE T2 BIT(8), EXCHANGES BIT(1);

      /* THE FIRST STEP IS TO SORT THE CODE FIXUP TABLE */
      K,FXLIM = FCP - 1;      EXCHANGES = TRUE;
      DO WHILE EXCHANGES;  /* QUIT BUBBLE SORT AFTER TABLE QUIETS DOWN */
         EXCHANGES = FALSE;  /* RESET ON EACH EXCHANGE BELOW */
         DO J = 0 TO K-1;
            I = FXLIM-J;
            L = I-1;
            IF FIXCADR(L) > FIXCADR(I) THEN
               DO;  /* SWAP */
                  T1 = FIXCADR(L);  FIXCADR(L) = FIXCADR(I);  FIXCADR(I) = T1;
                  T2 = FIXCB1(L);  FIXCB1(L) = FIXCB1(I);  FIXCB1(I) = T2;
                  T2 = FIXCB2(L);  FIXCB2(L) = FIXCB2(I);  FIXCB2(I) = T2;
                  EXCHANGES = TRUE;  K = J;
               END;
         END;
      END;

      /* NOW WRITE OUT THE CURRENT BLOCK */
      FILE(CODEFILE,CURCBLK) = CODE;

      /* WRITE BINARY PROGRAM PATCHES INTO PROGRAM FILE */

      K,PPORG=0;  PPLIM = DISKBYTES;
      DO J = 0 TO CODEMAX;

         I = K;  /* KEEP TRACK OF K SO THAT WE WILL KNOW WHEN TO READ IN */

         DO WHILE (K <= FXLIM)  &  (FIXCADR(K) < PPLIM);
            /* IF THE FILE HAS NOT YET BEEN READ IN, DO SO */
            IF K = I THEN CODE = FILE(CODEFILE,J); /* ONLY IF A FIX IS NEEDED */
            L = FIXCADR(K) - PPORG;  /* RELATIVE ADDRESS WITHIN THIS BLOCK */
            CODE(L) = FIXCB1(K);  CODE(L+1) = FIXCB2(K);
            K = K + 1;
         END;
```

```
      IF K > I THEN     /* A FIXUP WAS DONE */
         FILE(CODEFILE,J) = CODE;  /* SO WRITE OUT THE CONTENTS */

      PPORG = PPORG + DISKBYTES;
      PPLIM = PPLIM + DISKBYTES;
   END;

   FCP = 0;  /* RESET TABLE TO EMPTY */
   CODE = FILE(CODEFILE,CURCBLK);  /* RESTORE FILE TO PREVIOUS STATE */
   PPORG = CURCBLK*DISKBYTES;  PPLIM = PPORG + DISKBYTES;
END INSERT_CODE_FIXUPS;

FIXCHW:
   PROCEDURE (ADR, B1, B2);
      DECLARE ADR FIXED, (B1, B2) BIT(8);
      /*  FIX UP ONE HALF WORD OF CODE  */
      COUNT$FIXCHW = COUNT$FIXCHW + 1;
      IF FCP >= FCLIM THEN
         CALL INSERT_CODE_FIXUPS;
      IF PPORG <= ADR & ADR < PPLIM THEN
         DO;
            SHORTCFIX = SHORTCFIX + 1;
            ADR = ADR - PPORG;
            CODE(ADR) = B1;
            CODE(ADR+1) = B2;
         END;
      ELSE
         DO;
            LONGCFIX = LONGCFIX + 1;
            FIXCADR(FCP) = ADR;
            FIXCB1(FCP) = B1;
            FIXCB2(FCP) = B2;
            FCP = FCP + 1;
         END;
   END  FIXCHW;


FIXBFW:
   PROCEDURE (WHERE, VAL);
      DECLARE (WHERE, VAL, I, J, P) FIXED;
      IF WHERE = 0 THEN RETURN;
      /* FIX UP A BRANCH WHOSE ADDRESS WE NOW KNOW */
      COUNT$FIXBFW = COUNT$FIXBFW + 1;
      IF CONTROL(BYTE(CON_E)) THEN OUTPUT = X4 \\ X4 \\ WHERE \\ ': FIXUP ='
            \\ VAL;
      P = WHERE + 2;        /* THE ACTUAL ADDRESS FIELD  */
      IF WHERE >=  "1000" THEN
         DO;
            CALL FIXCHW (P, SHL(DBR,4), SHR(VAL,10) & "FC");
            VAL = VAL & "FFF";
            P = P + 4;
         END;
      ELSE IF VAL >= "1000" THEN
         DO;
            I = VAL & "FFF";
```

```
            J = SHR(VAL, 12);
            INSTRUCT(LOAD) = INSTRUCT(LOAD) + 1;
            INSTRUCT(BC) = INSTRUCT(BC) + 1;
            CALL EMITDATAWORD (SHL(LOAD, 24) + SHL(BRCHREG, 20) + SHL(DBR, 12)
                  + SHL(J, 2));
            CALL EMITDATAWORD("47F00000" +  SHL(BRCHREG,16) + SHL(PBR, 12) + I);
            CALL FINDADDRESS (DP-8);
            CALL FIXCHW(P, SHL(ADREG,4)+SHR(ADRDISP,8), ADRDISP & "FF");
            RETURN;
         END;
      CALL FIXCHW(P, SHL(PBR,4)+SHR(VAL,8), VAL & "FF");
   END FIXBFW;


FIXWHOLEDATAWORD:
   PROCEDURE (ADR, WORD);
      DECLARE (ADR, WORD) FIXED;
      DECLARE (BLK, TEMP) FIXED, REREAD BIT(1);
      IF CONTROL(BYTE(CON_E)) THEN
         OUTPUT = X4 \\ ADR \\ ':  FIXUP = ' \\ WORD;
      COUNT#FIXD = COUNT#FIXD + 1;
      BLK = ADR/DISKBYTES;
      REREAD = (CURDBLK ^= BLK);
      IF REREAD THEN
         DO;  /* MUST GET THE RIGHT BLOCK  */
            LONGDFIX = LONGDFIX + 1;
            TEMP = DP;
            DP = ADR;
            CALL GETDATA;
         END;
      ELSE SHORTDFIX = SHORTDFIX + 1;
      ADR = ADR MOD DISKBYTES;
      DATA(ADR) = SHR(WORD, 24);
      DATA(ADR+1) = SHR(WORD, 16);
      DATA(ADR+2) = SHR(WORD, 8);
      DATA(ADR+3) = WORD;
      IF REREAD THEN DP = TEMP;
   END  FIXWHOLEDATAWORD;


ENTER:
   PROCEDURE (N, T, L, LINE);
      /*  ENTER A SYMBOL IN THE SYMBOL TABLE  */
      DECLARE (I, J, K, L, T, LINE) FIXED, N CHARACTER;

      DO I = PROCMARK TO NDECSY;
         IF N = SYT(I) THEN
            DO;
               K = SYTYPE(I);
               IDCOMPARES = IDCOMPARES + I - PROCMARK;
               IF T = LABELTYPE & (K = FORWARDTYPE \ K = FORWARDCALL) THEN
                  DO;
                     IF CONTROL(BYTE(CON_E)) THEN
                        OUTPUT = X4 \\ 'FIX REFERENCES TO: ' \\ N;
                     J = BASES(SYBASE(I))+ SYDISP(I);
                     IF K = FORWARDCALL THEN
```

```
                        IF L > '0FFF' THEN
                           L = L+8;
                        ELSE
                           L = L+4;
                     SYBASE(I) = SHR(L, 12);
                     SYDISP(I) = L & '0FFF';
                     CALL FIXWHOLEDATAWORD(J,L);
                     SYTYPE(I) = T;
                     DECLARED_ON_LINE(I) = LINE;
                  END;
               ELSE IF PROCMARK + PARCT < I THEN
                  CALL ERROR('DUPLICATE DECLARATION FOR:  ' \\ N, 0);
               ELSE DECLARED_ON_LINE(I) = LINE;
               RETURN I;
            END;
      END;
      NDECSY = NDECSY + 1;
      IF NDECSY > MAXNDECSY THEN
         IF NDECSY > SYTSIZE THEN
            DO;
               CALL ERROR ('SYMBOL TABLE OVERFLOW', 1);
               NDECSY = NDECSY - 1;
            END;
         ELSE MAXNDECSY = NDECSY;
      SYT(NDECSY) = N;
      SYTYPE(NDECSY) = T;
      DECLARED_ON_LINE(NDECSY) = LINE;
      SYTCO(NDECSY) = 0;
      SYESDTYPE(NDECSY) = ESDTYPE;
      IF ESDTYPE ^= 0 & LENGTH(N) > 7 THEN
         CALL ERROR('EXTERNAL,PUBLIC, OR COMMON NAME EXCEEDS 7 CHARACTERS');
      IF T = LABELTYPE THEN
         DO;
            SYBASE(NDECSY) = SHR(L, 12);  /* PAGE  */
            SYDISP(NDECSY) = L & '0FFF';
         END;
      ELSE
         DO;
            CALL FINDADDRESS(L);
            SYBASE(NDECSY) = ADREG;
            SYDISP(NDECSY) = ADRDISP;
         END;
      IDCOMPARES = IDCOMPARES + NDECSY - PROCMARK;
      RETURN NDECSY;
   END  ENTER;

ID_LOOKUP:
   PROCEDURE (P);
      /* LOOKS UP THE IDENTIFIER AT P IN THE ANALYSIS STACK IN THE
         SYMBOL TABLE AND INITIALIZES FIXL,CNT,TYPE,REG,INX
         APPROPRIATELY.  IF THE IDENTIFIER IS NOT FOUND, FIXL IS
         SET TO -1
      */
      DECLARE P FIXED, I FIXED;
      CHAR_TEMP = VAR(P);
      DO I = 0 TO NDECSY - 1;
```

```
            IF SYT(NDECSY-I) = CHAR_TEMP THEN
              DO;
                 IDCOMPARES = IDCOMPARES + I;
                 I,FIXL(P) = NDECSY - I;
                 CNT(P) = 0;    /* INITIALIZE SUBSCRIPT COUNT  */
                 TYPE(P) = VARIABLE;
                 IF SYTYPE(I) = SPECIAL THEN
                    FIXV(P) = SYDISP(I);          /* BUILTIN FUNCTION */
                 ELSE
                    FIXV(P) = 0;                  /* ~ BUILTIN FUNCTION */
                 REG(P),INX(P) = 0;    /* INITIALIZE REGISTER POINTERS */
                 SYTCO(I) = SYTCO(I) + 1;  /* COUNT REFERENCES */
                 RETURN;
              END;
         END;
         IDCOMPARES = IDCOMPARES + NDECSY;
         FIXL(P) = -1;       /*  IDENTIFIER NOT  FOUND */
      END   ID_LOOKUP;


   UNDECLARED_ID:
      PROCEDURE (P);
         /* ISSUES AN ERROR MESSAGE FOR UNDECLARED IDENTIFIERS AND
            ENTERS THEM WITH DEFAULT TYPE IN THE SYMBOL TABLE
         */
         DECLARE P FIXED;
         CALL ERROR('UNDECLARED IDENTIFIER:  ' \\ VAR(P) ,0);
         CALL EMITDATAWORD(0);
         CALL ENTER (VAR(P), FIXEDTYPE, DP-4, CARD_COUNT);
         CNT(P) = 0;
         FIXV(P) = 0;
         REG(P) = 0;
         INX(P) = 0;
         FIXL(P) = NDECSY;
         SYTCO(NDECSY) = 1;                  /* COUNT FIRST REFERENCE */
         TYPE(P) = VARIABLE;
      END  UNDECLARED_ID;


    /* PROCEDURES RELATED TO PUBLIC AND EXTERNAL PROCEDURE CALLS */

    /* MAJOR WEST VIRGINIA UNIVERSITY ENHANCEMENTS FOLLOW        */

   BUILD_PROLOGUE:
      PROCEDURE(SYPTR);  /* CREATE CODE FOR BEGINNING OF "PUBLIC" PROC */
         DECLARE (SYPTR,I,J) FIXED;
         DECLARE (B1,B2) BIT(8);

         PP = (PP + 3) & "FFFFFC"; /* SET PP TO FULL WORD BOUNDRY */
         SYBASE(SYPTR) = SHR(PP,12);  /* SET BASE OF PROCEDURE */
         SYDISP(SYPTR) = PP & "FFF";  /* DISPLACEMENT */
         CALL EMITRX(BC,15,0,CALLREG,16);   /* B * + 12  */
         IF LENGTH(CURRENT_PROCEDURE) < 8 THEN
            S = PAD(CURRENT_PROCEDURE,8);
         ELSE
            S = CURRENT_PROCEDURE;
```

```
      BYTE(S,0) = LENGTH(CURRENT_PROCEDURE) - 1;
      DO I = 0 TO 7 BY 2;        /* INSERT NAME INTO CODE AREA */
         B1 = BYTE(S,I);
         B2 = BYTE(S,I+1);
         CALL EMITCODEBYTES(B1,B2);
      END;
      SYRLDADDR(SYPTR) = PP;          /* ADDR OF DATA CSECT GOES HERE */
      PP = PP + 4;


   /* NOW EMIT CODE TO SAVE THE CALLERS REGISTERS VIA IBM/OS CONVENTIONS */

      CALL EMITRX(STM,BRCHREG,PBR,SBR,12); /* SAVE R14 - R12 */
      CALL EMITRX(LOAD,DBR,0,CALLREG,12); /* LOAD ADDR OF DATA CSECT */
                                     /* R15 IS BASE BECAUSE WE ARE ENVOLKED*/
                                     /* VIA A   BALR R14,R15 INSTRUCTION */
      CALL EMITRX(LM,4,12,DBR,BASEDATA+16);   /* LOAD OUR R4 - R12  */

                                     /* R4-R11 = DATA BASE REG'S */
                                     /* R12 NOW BECOMES THE BASE REG */
      CALL EMITRX(STORE,SBR,0,DBR,SAVE+4); /* R13 TO OUR SAVE AREA */
      CALL EMITRX(LA,CALLREG,0,DBR,SAVE);   /* GET ADDR OF OUR SAVE AREA */
      CALL EMITRX(STORE,CALLREG,0,SBR,8);       /* INTO CALLER'S SAVE AREA */
      CALL EMITRX(LM,SBR,CALLREG,DBR,BASEDATA+52);  /* LOAD OUR R13 - R15 */
      CALL EMITDATAWORD(0);   /* SAVE AREA FOR PARM ADDRESS */
      PARMLOC = DP-4;
      CALL FINDADDRESS(PARMLOC);
      CALL EMITRX(STORE,0,0,ADREG,ADRDISP);  /* SAVE ADDRESS OF PARMS */
                                        /* NOTE R13 IS NOT THE SAVE ADDR*/

END BUILD_PROLOGUE;

BUILD_EXTERNAL_CALL:
   PROCEDURE(I);         /* SET UP FOR CALL TO PUBLIC PROC */
                      /* TRIGGERED BY "EXTERNAL: NAME PROCEDURE ..." */
      DECLARE(I,TREG,TDISP) FIXED;

      CALL EMITDATAWORD(0);
      CALL FINDADDRESS(DP-4);
      TREG = ADREG;          /* R13 SAVE AREA */
      TDISP = ADRDISP;
      CALL EMITRX(STORE,SBR,0,TREG,TDISP);
      CALL FINDADDRESS(DP);    /* REGISTER SAVE AREA FOR EXTERNAL CALL */
      DP = DP + 72;            /* 18 WORD SAVE AREA */
      CALL CHECKBASES;
      CALL EMITRX(LA,SBR,0,ADREG,ADRDISP);  /* CALLING CONVENTION */
      CALL EMITDATAWORD(0);                 /* ADDR OF EXTERNAL GOES HERE*/
      SYRLDADDR(I) = DP-4;
      CALL FINDADDRESS(DP-4);
      CALL EMITRX(LOAD,CALLREG,0,ADREG,ADRDISP);  /* GET REAL ADDR OF PROC */
      CALL EMITRX(LA,0,0,SYBASE(PROCMARK),SYDISP(PROCMARK)); /* PARM LIST ADDR*/
      CALL EMITRR(BALR,BRCHREG,CALLREG);

      CALL EMITRX(LOAD,SBR,0,TREG,TDISP); /* RESTORE R13 */

END BUILD_EXTERNAL_CALL;
```

```
RESTORE_REGISTERS:
   PROCEDURE;       /* RESTORE R4-R15 AFTER RETURN FROM AN EXTERNAL PROC */
      CALL EMITRX(LOAD,SBR,0,DBR,SAVE+4);
      CALL EMITRX(LM,14,15,13,12);        /* RESTORE R14 - R15 */
      CALL EMITRX(LM,4,12,13,36);         /* RESTORE R4 - R12 */
      CALL EMITRR("18",15,3);          /* RETURN CODE TO R15 FOR OS */
      CALL EMITRR(BCR,15,BRCHREG);        /* RETURN TO CALLER */
END RESTORE_REGISTERS;

EMIT_RETURN:
   PROCEDURE;    /* RETURN FROM EITHER A LOCAL OR EXTERNAL PROC */
      IF SYESDTYPE(PROCMARK-1) = "01" THEN
         CALL RESTORE_REGISTERS;  /* RETURNING FROM A PUBLIC PROC */
      ELSE
         DO;
            CALL FINDADDRESS(RTNADR);
            CALL EMITRX(LOAD,BRCHREG,0,ADREG,ADRDISP);
            CALL EMITRR(BCR,"F",BRCHREG);
         END;
END EMIT_RETURN;

SETUP_EXTRN:
   PROCEDURE(I);

 /* PROCEDURE TO INITIALIZE CODE FOR EITHER A EXTERNAL PROC OR
                                                 PUBLIC PROC */

      DECLARE(I) FIXED;


      DO CASE ESDTYPE;
         ;              /* CASE 00 - SECTION DEFINITION */
         DO;            /* CASE 01 - LABEL (PUBLIC ENTRY) DEFINITION */
            IF MAIN_PTR = 0 THEN
               MAIN_PTR = I;     /* FIRST PUBLIC PROCEDURE */
            CALL BUILD_PROLOGUE(I);
         END;

         DO;            /* CASE 02 - EXTERNAL REFERENCE    */
            CALL BUILD_EXTERNAL_CALL(I);
            CALL EMIT_RETURN;
         END;

       END;  /* DO CASE   */
       ESDTYPE = 0;  /*RESET FOR PARAMETER LIST */

END SETUP_EXTRN;

SETINIT:
   PROCEDURE;
      /*  PLACES INITIAL VALUES INTO DATA AREA */

      DECLARE (I, J) FIXED;

  /* CHECK FOR ATTEMPTED INITIALIZATION OF AN EXTERNALLY DEFINED VARIABLE */
```

```
      IF SYESDTYPE(FIXL(MP)) = "05" THEN
         DO;
            CALL ERROR('ILLEGAL INITIALIZATION OF COMMON VARIABLE',1);
            RETURN;
         END;

      IF ITYPE = CHRTYPE THEN
         DO;
            IF TYPE(MPP1) ^= CHRTYPE THEN VAR(MPP1) = FIXV(MPP1);
            S = VAR(MPP1);     /* THE STRING */
            I = LENGTH(S) - 1;

            IF I < 0 THEN
               CALL EMITDESC(0);
            ELSE
               CALL EMITDESC(SHL(I,24) + CHP);

            DO J = 0 TO I;
               CALL EMITCHAR(BYTE(S,J));
            END;
         END;
      ELSE IF TYPE(MPP1) ^= CONSTANT THEN
         CALL ERROR ('ILLEGAL CONSTANT IN INITIAL LIST');
      ELSE IF ITYPE = FIXEDTYPE THEN
         CALL EMITDATAWORD(FIXV(MPP1));
      ELSE IF ITYPE = HALFWORD THEN
         DO;
            /*  FIRST FORCE ALIGNMENT  */
            DP = (DP + 1) & "FFFFFE";
            CALL EMITBYTE (SHR(FIXV(MPP1), 8));
            CALL EMITBYTE(FIXV(MPP1) & "FF");
         END;
      ELSE IF ITYPE = BYTETYPE THEN
         CALL EMITBYTE(FIXV(MPP1));
   END  SETINIT;

ALLOCATE :
   PROCEDURE(P,DIM);
      /* ALLOCATES STORAGE FOR THE IDENTIFIER AT P IN THE ANALYSIS
         STACK WITH DIMENSION DIM
      */

      DECLARE (P, DIM, J, LEN) FIXED;


   CHECK_NEWDP:
      PROCEDURE;
         DECLARE T FIXED;
         T = DP;
         DP = NEWDP;
         CALL CHECKBASES;
         DP = T;
      END  CHECK_NEWDP;
```

```
SYTYPE(FIXL(P)) = TYPE(P);
IF SYESDTYPE(FIXL(P)) = EXT_PARM THEN  /* ALLOCATE ALREADY DONE*/
   DO;
      SYESDTYPE(FIXL(P)) = 0;
      RETURN;
   END;
IF ESDTYPE = "05" THEN  /* ALIGN TO WORD BOUNDRY FOR ADDR */
   DO;
      NEWDP = (NEWDP + 3) & "FFFFFC";
      CALL CHECK_NEWDP;
   END;

DIM = DIM + 1;          /* ACTUAL NUMBER OF ITEMS  */
DO CASE TYPE(P);

   ;     /*  CASE  0    DUMMY          */

   DO;       /*   CASE 1    HALFWORD  */
      NEWDP = (NEWDP + 1) & "FFFFFE";   /* ALIGN HALFWORD  */
      CALL CHECK_NEWDP;
      J = NEWDP;
      LEN = SHL(DIM, 1);
   END;


   ;     /*  CASE  2    LABEL TYPE          */


    ;    /*  CASE  3    ACCUMULATOR         */

    ;    /*  CASE  4    VARIABLE            */

    ;    /*  CASE  5    CONSTANT            */

    ;    /*  CASE  6    CONDITION           */

    DO;  /*  CASE  7    CHARACTER TYPE      */
      IF ESDTYPE ^= "05" THEN  /* NOT AN EXTERNAL */
         DO;
            J = -NEWDSP;
            NEWDSP = NEWDSP + SHL(DIM,2);
            LEN = 0;
         END;
      ELSE
         DO; /* A COMMON STRING IS BEING DECLARED */
            CALL EMITDATAWORD(0);
            J = NEWDP;
            LEN = SHL(DIM,2);  /* NUMBER OF DESCRIPTORS */
         END;

    END;

    DO;  /*  CASE  8    FIXED TYPE          */
      NEWDP = (NEWDP + 3) & "FFFFFC";    /* ALIGN TO WORD */
      CALL CHECK_NEWDP;
      J = NEWDP;
```

```
      LEN = SHL(DIM,2);
    END;

    DO;  /*  CASE  9     BYTE TYPE           */
      CALL CHECK_NEWDP;
      J = NEWDP;
      LEN = DIM;
    END;

   DO;  /*  CASE 10     FORWARD TYPE  (LABEL)  */
      NEWDP = (NEWDP+3) & "FFFFFC";  /* WORD ALIGN */
      CALL CHECK_NEWDP;
      J = NEWDP;
      LEN = SHL(DIM,2);          /* SPACE FOR FIXUPS  */
   END;

    ;     /*  CASE 11     DESCRIPT            */

    ;     /*  CASE 12     SPECIAL             */

    ;     /*  CASE 13     FORWARD CALL        */

    ;     /*  CASE 14     CHAR_PROC_TYPE                */

    ;     /*  CASE 15     UNUSED              */

END; /*  OF DO CASE TYPE(P)  */

CALL FINDADDRESS(J);
SYBASE(FIXL(P)) = ADREG;
SYDISP(FIXL(P)) = ADRDISP;
IF ESDTYPE ^= "05" THEN
   NEWDP = NEWDP + LEN;
ELSE
   DO;
      SYLEN(FIXL(P)) = LEN;
      SYRLDADDR(FIXL(P)) = J;
      NEWDP = NEWDP + 4; CALL CHECK_NEWDP;
   END;
IF SYESDTYPE(FIXL(P)) = "0E" THEN  /* MOVE EXTERNAL PARM */
   DO;
      CALL FINDADDRESS(PARMLOC);
      J = FINDAC;
      CALL EMITRX(LOAD,J,0,ADREG,ADRDISP); /* BASE ADDR */
      CALL EMITRX(LA,J,0,J,SHL(SYTCO(FIXL(P))-1,2)); /* PARM ADDR*/
      CALL EMITRR("18",15,J);  /*SAVE FOR CLEAR */
      CALL EMITRX(LOAD,J,0,J,0); /* PARM VALUE */
      CALL EMITRR("1B",BRCHREG,BRCHREG); /*ZERO VALUE TO */
      CALL EMITRX(STORE,BRCHREG,0,15,0); /*CLEAR PARM FIELD*/
      IF TYPE(P) = CHRTYPE THEN
         DO;
            CALL EMITRX(STORE,J,0,DBR,STRN); /*INCOMMING DESCRIPTOR*/
            CALL EMITRX(STORE,SBR,0,DBR,STRREG);
            CALL EMITRX(BAL,BRCHREG,0,PBR,STRMOVE);
            CALL EMITRR("18",J,3);
         END;
```

```
            IF SYTYPE(FIXL(P)) = BYTETYPE THEN
               DIM = "42";              /*  STC  */
            ELSE IF SYTYPE(FIXL(P)) = HALFWORD THEN
               DIM = "40";
            ELSE
               DIM = STORE;             /*  ST  */
            CALL EMITRX(DIM, J, 0, SYBASE(FIXL(P)), SYDISP(FIXL(P)));
            BASES(J) = AVAIL;
            SYTCO(FIXL(P)),SYESDTYPE(FIXL(P)) = 0;
         END;
      SYESDTYPE(FIXL(P)) = ESDTYPE; /* SET TO "05" IF EXTERNAL ELSE "00"*/


   END  ALLOCATE;


TDECLARE:
   PROCEDURE (DIM);
      /*  ALLOCATES STORAGE FOR IDENTIFIERS IN DECLARATIONS  */
      DECLARE DIM FIXED;
      NEWDP = DP;
      NEWDSP = DSP;
      TYPE(MP) = TYPE(SP);
      CASEP = FIXL(MP);
      DO I = 1 TO INX(MP);
         FIXL(MP) = CASESTACK(CASEP+I);      /* SYMBOL TABLE POINTER */
         CALL ALLOCATE(MP, DIM);
      END;
   END  TDECLARE;



MOVESTACKS:
   PROCEDURE (F,T);
      DECLARE F FIXED, T FIXED;
      /*  MOVE ALL THE COMPILER STACKS DOWN FROM F TO T  */
      TYPE(T) = TYPE(F);  VAR(T) = VAR(F);
      FIXL(T) = FIXL(F);  FIXV(T) = FIXV(F);
      INX(T) = INX(F);    REG(T) = REG(F);
      PPSAVE(T) = PPSAVE(F);  CNT(T) = CNT(F);
   END  MOVESTACKS;



  /*                          BRANCH PROCEDURES                          */


BRANCH_BD:
   PROCEDURE(COND, B, D);
      DECLARE (COND, B, D) FIXED;
      /*  BRANCHES ARE A SPECIAL CASE.  IF THEY ARE INTO THE 1ST  4096
         BYTES OF PROGRAM A SINGLE BRANCH WILL SUFFICE.  OTHERWISE WE
         MUST INDEX WITH A CONSTANT IN BRCHREG TO GET ANYWHERE.
```

```
      */
      IF B = 0 THEN
         CALL EMITRX(BC, COND, 0, PBR, D);
      ELSE
         DO;
            CALL EMITRX(LOAD,BRCHREG,0,DBR,SHL(B,2));
            CALL EMITRX(BC, COND, BRCHREG, PBR, D);
         END;
   END  BRANCH_BD;


BRANCH:
   PROCEDURE (COND, LOCATION);
      DECLARE (COND, LOCATION) FIXED;
      IF LOCATION = 0 THEN LOCATION = PP;
      /* ASSUME FIXUP WILL BE NEAR  */
      CALL BRANCH_BD(COND, SHR(LOCATION,12), LOCATION & "FFF");
   END BRANCH;



  /*                    EXPRESSIONS                                  */


CONDTOREG:
   PROCEDURE (MP, CC);
      DECLARE (MP, CC, J) FIXED;
      J = FINDAC;
      CALL EMITRX(LA, J, 0, 0, 0);
      IF PP < 4084 THEN
         CALL BRANCH(CC, PP+8);
      ELSE
         CALL BRANCH(CC, PP+12);
      CALL EMITRX(LA, J, 0, 0, 1);
      TYPE(MP) = ACCUMULATOR;
      REG(MP) = J;
      STILLCOND = CC;
   END CONDTOREG;

BRLINK_BD:
   PROCEDURE (BASE,DISP);
      DECLARE (BASE, DISP) FIXED;
      IF BASE = 0 THEN
         CALL EMITRX(BAL, BRCHREG, 0, PBR, DISP);
      ELSE
         DO;
            CALL EMITRX(LOAD, BRCHREG, 0, DBR, SHL(BASE,2));
            CALL EMITRX(BAL, BRCHREG, BRCHREG, PBR, DISP);
         END;
   END  BRLINK_BD;

   /*                  CODE FOR PROCEDURES                           */
```

```
SAVE_REGISTERS:
   PROCEDURE;
      /* GENERATES CODE TO SAVE REGISTERS BEFORE A PROCEDURE OR
         FUNCTION CALL
      */
      DECLARE I FIXED;
      DO I = 1 TO 3;
         IF BASES(I) ^= AVAIL THEN
            DO;
               CALL EMITDATAWORD(0);
               CALL FINDADDRESS(DP-4);
               TEMP(I) = SHL(ADREG,12)+ ADRDISP;
               CALL EMITRX(STORE,I,0,ADREG,ADRDISP);
            END;
         ELSE
            TEMP(I) = 0;
      END;
   END  SAVE_REGISTERS;


UNSAVE_REGISTERS:
   PROCEDURE (R,P);
      /*  GENERATES CODE TO RESTORE REGISTERS AFTER A FUNCTION
          OR PROCEDURE CALL AND ALSO DOES SOME HOUSEKEEPING
      */
      DECLARE (R, P, I, J) FIXED;
      IF BASES(R) ^= AVAIL THEN
         DO;
            J = FINDAC;
            CALL EMITRR('18', J, R);
         END;
      ELSE
         J = R;
      DO I = 1 TO 3;
         IF TEMP(I) ^= 0 THEN
            CALL EMITRX(LOAD,I,0,SHR(TEMP(I),12), TEMP(I)&"FFF");
      END;
      TYPE(P) = ACCUMULATOR;
      REG(P) = J;
      BASES(J) = ACCUMULATOR;
   END  UNSAVE_REGISTERS;



CALLSUB:
   PROCEDURE (SB,SD, R, P);
      DECLARE (SB, SD, R, P) FIXED;
      CALL SAVE_REGISTERS;
      CALL BRLINK_BD(SB,SD);
      CALL UNSAVE_REGISTERS(R, P);
   END  CALLSUB;


CALLSUB_FORWARD:
   PROCEDURE (SB,SD,R,P);
```

```
      DECLARE (SB, SD, R, P) FIXED;
      CALL SAVE_REGISTERS;
      CALL EMITRX(LOAD, BRCHREG, 0, SB,SD);
      CALL EMITRX(BAL, BRCHREG, BRCHREG, PBR, 0);
      CALL UNSAVE_REGISTERS(R, P);
   END  CALLSUB_FORWARD;


FORCE_ADDRESS:
   PROCEDURE (SP,R);
      /* GENERATES THE ADDRESS OF THE <VARIABLE> IN THE ANALYSIS
         STACK AT SP IN REGISTER R.
      */
      DECLARE (SP, R, K, INXSP) FIXED;
      IF SYTYPE(FIXL(SP)) = LABELTYPE THEN
         DO;
            K = FIXL(SP);
            IF SYBASE(K) = 0 THEN
               CALL EMITRX(LA,R,0,PBR,SYDISP(K));
            ELSE
               DO;
                  CALL EMITRX(LOAD,R,0,DBR,SHL(SYBASE(K),2));
                  CALL EMITRX(LA,R,R,PBR,SYDISP(K));
               END;
         END;
      ELSE
         DO;
            K = SYTYPE(FIXL(SP));
            INXSP = INX(SP);
            IF INXSP ^= 0 THEN
               DO;
                  IF K ^= BYTETYPE THEN
                     IF K = HALFWORD THEN
                        CALL EMITRR ("1A", INXSP, INXSP);
                     ELSE
                        CALL EMITRX("89",INXSP,0,0,2);
                  BASES(INXSP) = AVAIL;
               END;
            IF K = FORWARDTYPE \ K = FORWARDCALL THEN
               DO;
                  K = FIXL(SP);
                  CALL EMITRX(LOAD,R,0,SYBASE(K),SYDISP(K));
                  CALL EMITRR("1A",R,PBR);
               END;
            ELSE
               CALL EMITRX(LA,R,INXSP,SYBASE(FIXL(SP)),SYDISP(FIXL(SP)));
         END;
   END  FORCE_ADDRESS;


FILE_PSEUDO_ARRAY:
   PROCEDURE (VARP,FILEP, DIRECTION);
      /* PROCEDURE TO GENERATE CODE FOR THE FILE PSEUDO ARRAY.
         TWO FORMS ARE HANDLED:
```

```
            <VARIABLE>  =  FILE(I,J);

            FILE(I,J)  =  <VARIABLE>;

         VARP IS A POINTER TO THE <VARIABLE> IN THE ANALYSIS STACKS.
         FILEP IS A POINTER TO THE ANALYSIS STACK WHERE FILE(I,J)
         HAS BEEN ASSIMILATED UNDER THE GUISE OF A SUBSCRIPTED
         VARIABLE.  DIRECTION = 0 FOR THE FIRST CASE (READ) AND
         DIRECTION = 4 FOR THE SECOND CASE (WRITE).  I IS THE FILE
         INDEX (I = 1,2,3) AND J IS THE RELATIVE RECORD WITHIN THE
         FILE.  THE GENERATED CODE SHOULD HAVE THE SAME EFFECT AS;

            LA    0,<VARIABLE>
            L     1,I
            SLL   1,3                    I*8
            LA    1,DIRECTION+44(,1)
            L     2,J
            L     BRCHREG,XPLMON         V_CON FOR THE MONITOR
            BALR  BRCHREG,CALLREG

         REGISTERS 0-3 ARE NOT PRESERVED ACROSS THE MONITOR CALL,
         HENCE ALL REGISTERS ARE FREED,

      */
      DECLARE (VARP, DIRECTION, FILEP, R) FIXED;
      IF TYPE(VARP) = VARIABLE THEN
         DO;
            CALL FORCE_ADDRESS(VARP,0);
            CALL EMITRX("89",REG(FILEP),0,0,3);    /*  I*8  */
            R = FINDAC;
            IF INX(FILEP) = 1 THEN
               DO;                               /*  JUGGLE REGISTERS  */
                  CALL EMITRR("18",R,1);
                  INX(FILEP) = R;
               END;
            CALL EMITRX(LA,1,0,REG(FILEP),44+DIRECTION);
            IF INX(FILEP) ^= 2 THEN
               CALL EMITRR("18",2,INX(FILEP));     /*  J    */
            CALL EMITRX(LA,SBR,0,DBR,SAVE); /* SAVE AREA ADDR */
            CALL EMITRX(LOAD,CALLREG,0,DBR,XPLMON);
            CALL EMITRR(BALR,BRCHREG,CALLREG);
            CALL EMITRX(LOAD,SBR,0,DBR,BASEDATA+52); /* RESTORE R13 */
            TYPE(FILEP) = SPECIAL;          /*  NO MORE ASSIGNMENTS  */
            CALL CLEARREGS;                /*  FREE ALL REGISTERS  */
         END;
      ELSE
         CALL ERROR('ILLEGAL USE OF FILE PSEUDO ARRAY',1);
   END  FILE_PSEUDO_ARRAY;


EMIT_INLINE:
   PROCEDURE;

      /* GENERATES CODE FOR THE PSEUDO FUNCTION INLINE                  */

      DECLARE BINLM CHARACTER INITIAL ('BAD ARGUMENT TO INLINE');
```

```
      IF CNT(MP) < 4 THEN
         DO;
            IF TYPE(MPP1) = CONSTANT THEN
               DO CASE CNT(MP);

                  ;                                  /* NO CASE 0 */

                  FIXL(MP) = FIXV(MPP1);             /* SAVE OP CODE */

                  DO;                                /* SAVE R1  */
                     TYPE(MP) = ACCUMULATOR;
                     REG(MP) = FIXV(MPP1);
                  END;

                  CALL EMITCODEBYTES(FIXL(MP), SHL(REG(MP), 4) +
                     FIXV(MPP1));
                                                     /* EMIT  OP R1 X  */
               END;
            ELSE
               CALL ERROR(BINLM,1);
         END;
      ELSE IF TYPE(MPP1) = CONSTANT THEN
         DO;
            IF CNT(MP) & 1 THEN
               CALL EMITCODEBYTES(INX(MP)+SHR(FIXV(MPP1), 8),
                  FIXV(MPP1));
                                          /* EMIT  B DDD  */
            ELSE
               INX(MP) = SHL(FIXV(MPP1), 4);      /* SAVE BASE REG  */
         END;
      ELSE IF TYPE(MPP1) = VARIABLE THEN
         DO;
            CNT(MP) = CNT(MP) + 1;
            IF CNT(MP) & 1 THEN
               CALL EMITCODEBYTES(SHL(SYBASE(FIXL(MPP1)), 4) +
                                  SHR(SYDISP(FIXL(MPP1)), 8) ,
                                  SYDISP(FIXL(MPP1)));
            ELSE
               CALL ERROR(BINLM, 1);
         END;
      ELSE
         CALL ERROR(BINLM, 1);

   END  EMIT_INLINE;



PROC_START:
   PROCEDURE;
      /*  GENERATES CODE FOR THE HEAD OF A PROCEDURE */

      DECLARE I FIXED;
      I = FIXL(MP);
      FIXL(MP) = PP;
```

```
      CALL BRANCH('F',0);     /*  BRANCH AROUND  */
      SYBASE(I) = SHR(PP,12);          /* ADDRESS OF THE PROCEDURE  */
      SYDISP(I) = PP & 'FFF';
      SYESDTYPE(I) = ESDTYPE; /* SET TYPE TO INTERNAL,PUBLIC, OR EXTERNAL */
      CALL EMITDATAWORD(0);  /*  PLACE TO STORE RETURN ADDRESS */
      RTNADR = DP - 4;
      PPSAVE(MP) = RTNADR;
      CALL FINDADDRESS(RTNADR);
      CALL EMITRX(STORE, BRCHREG,0,ADREG,ADRDISP);
      IF ESDTYPE ~= '00' THEN
            CALL SETUP_EXTRN(I);   /* EXTERNAL OR PUBLIC PROC */
   END  PROC_START;


STUFF_PARAMETER:
   PROCEDURE;
      /* GENERATES CODE TO SEND AN ACTUAL PARAMETER TO A PROCEDURE */
      I = FIXL(MP) + CNT(MP);
      IF LENGTH(SYT(I)) = 0 THEN
         DO;
            IF SYTYPE(I) = BYTETYPE THEN
               J = '42';            /*  STC  */
            ELSE IF SYTYPE(I) = HALFWORD THEN
               J = '40';
            ELSE
               J = STORE;          /*  ST  */
            CALL EMITRX(J, REG(MPP1), 0, SYBASE(I), SYDISP(I));
            BASES(REG(MPP1)) = AVAIL;
         END;
      ELSE
         CALL ERROR('TOO MANY ACTUAL PARAMETERS', 1);
   END  STUFF_PARAMETER;


CHECK_STRING_OVERFLOW:
   PROCEDURE;
      DECLARE I FIXED;
      CALL EMITRX (LOAD, 0, 0, SBR, FREEPOINT);
      CALL EMITRX (CMPR, 0, 0, SBR, FREELIMIT);
      I = PP;
      CALL BRANCH (4, 0);
      CALL EMITRR('18',0,SBR);  /* STRING TO BE COMPACTED */
      CALL EMITRX(STM,0,3,DBR,BASEDATA);
      CALL EMITRX(STORE,BRCHREG,0,DBR,SAVE); /*SAVE RETURN REG*/
      CALL EMITRX(LA,SBR,0,DBR,SAVE);  /* LINK SAVE AREA */
      CALL EMITRX(LOAD,CALLREG,0,DBR,XPLCOMP);
      CALL EMITRR(BALR,BRCHREG,CALLREG);
      CALL EMITRX(LM,0,3,DBR,BASEDATA);
      CALL EMITRR('18',SBR,0);    /* RESTORE R13 TO ORIGINAL VALUE */
      CALL EMITRX(LOAD,BRCHREG,0,DBR,SAVE); /*RESTORE RETURN ADDR*/
      CALL FIXBFW (I, PP);
   END CHECK_STRING_OVERFLOW;


FORCEACCUMULATOR:
   PROCEDURE (P);
```

```
DECLARE P FIXED;
/* FORCE THE OPERAND AT P INTO AN ACCUMULATOR */
DECLARE (R, SB, SD, TP, SFP, I) FIXED, T1 CHARACTER;

CHECK_COMMON:
   PROCEDURE;  /* LOAD DATA FROM COMMON IF NECESSARY */
      DECLARE OFFSET FIXED;
      IF SYESDTYPE(I) ^= "05" THEN   /* NOT COMMON */
         DO;
            CALL EMITRX(TP,R,INX(P),SB,SD);
            RETURN;
         END;
      CALL EMITRX(LOAD,BRCHREG,0,SB,SD); /* ADDR OF DATA */
      IF SFP = CHRTYPE  THEN /* INC PAST COM PREAMBLE*/
         OFFSET = 20;   /*OFFSET FOR STRING PREFIX */
      ELSE  OFFSET = 0;
      CALL EMITRX(TP,R,INX(P),BRCHREG,OFFSET); /* GET THE DATA */
END CHECK_COMMON;

COUNT$FORCE = COUNT$FORCE + 1;
TP = TYPE(P);
SB = SYBASE(FIXL(P));
I = FIXL(P);  /* SYMBOL TABLE POINTER */
SD = SYDISP(I);
SFP = SYTYPE(I);
IF TP = CONDITION THEN CALL CONDTOREG (P, REG(P));
ELSE IF TP = VARIABLE THEN
   DO;
      IF SFP = LABELTYPE \ SFP = CHAR_PROC_TYPE THEN
         DO;
            CALL CALLSUB(SB,SD,3,P);
            IF LENGTH(SYT(FIXL(P)+CNT(P)+1)) = 0 THEN
               IF CONTROL(BYTE('N')) THEN
                  OUTPUT = '** WARNING--NOT ALL PARAMETERS SUPPLIED';
            IF SFP = CHAR_PROC_TYPE THEN
               TYPE(P) = DESCRIPT;
         END;
      ELSE IF SFP = FORWARDTYPE \ SFP = FORWARDCALL THEN
         DO;
            CALL CALLSUB_FORWARD(SB,SD,3,P);
            SYTYPE(FIXL(P)) = FORWARDCALL;
         END;
      ELSE  IF SFP = SPECIAL THEN
         DO;
            CALL EMITRX("90", 1, 3, DBR, BASEDATA+4);
            IF SD = 6 THEN
               DO;  /*  INPUT */
                  CALL CHECK_STRING_OVERFLOW;
                  IF REG(P) = 0 THEN CALL EMITRR ("1B", 2, 2);
                  ELSE IF REG(P)^=2 THEN CALL EMITRR("18", 2, REG(P));
                  BASES(REG(P)) = AVAIL;
                  CALL EMITRX (LOAD, 0, 0, SBR, FREEPOINT);
                  /* THIS IS A POINTER TO THE FIRST FREE STRING AREA*/
                  CALL EMITRX (LA, 1, 0, 0, 4); /* 4 IS READ CARD  */
                  CALL EMITRX(LA,SBR,0,DBR,SAVE);
                  CALL EMITRX(LOAD,CALLREG,0,DBR,XPLMON);
```

```
            CALL EMITRR (BALR, BRCHREG, CALLREG); /* MONITOR CALL*/
            /* MOVE FREE STRING AREA POINTER */
            CALL EMITRX(LOAD,SBR,0,DBR,BASEDATA+52);
            CALL EMITRX (STORE, 1, 0, SBR, FREEPOINT);
            CALL EMITRX (STORE, 0, 0, DBR, STRL);
            REG(P) = 0;
            TYPE(P) = DESCRIPT;
         END;
      ELSE IF SD = 8 THEN
         CALL FILE_PSEUDO_ARRAY(P-2,P,0);
      ELSE IF SD >= 11 & SD <= 18 THEN
         DO;
            /* TRACE, UNTRACE, EXIT, TIME, DATE, ETC.          */
            IF SD = 15 THEN R = 1; ELSE R = 0;
            IF SD > 15 THEN
               DO;
                  IF REG(P) ^= 0 THEN
                     CALL EMITRR ("18", 0, REG(P));
                  BASES(REG(P)) = AVAIL;
                  IF INX(P) ^= 2 THEN
                     CALL EMITRR ("18", 2, INX(P));
                  BASES(INX(P)) = AVAIL;
               END;
            /* SET UP MONITOR REQUEST CODE */
            CALL EMITRX(LA,1,0,0,SHL(SD-R,2)-32);
            /* MONITOR CALL */
            CALL EMITRX(LA,SBR,0,DBR,SAVE);
            CALL EMITRX(LOAD,CALLREG,0,DBR,XPLMON);
            CALL EMITRR (BALR, BRCHREG, CALLREG);
            CALL EMITRX(LOAD,SBR,0,DBR,BASEDATA+52);
            TYPE(P) = ACCUMULATOR;
            IF R ^= 0 THEN
               CALL EMITRR ("18", 0, R);
            REG(P) = 0;
         END;
      ELSE CALL ERROR (' ILLEGAL USE OF ' \\ SYT(FIXL(P)));
      CALL EMITRX ("98", 1, 3, DBR, BASEDATA+4);
   END;
ELSE
   DO;  /* FETCH THE VARIABLE (ALL ELSE HAS FAILED) */
      IF SFP ^= BYTETYPE THEN
         DO;
            IF INX(P) ^= 0 THEN
              DO;
               IF SFP = HALFWORD THEN
                  CALL EMITRR ("1A", INX(P), INX(P));
               ELSE
                  CALL EMITRX ("89", INX(P), 0, 0, 2);
               /* SHIFT INDEX FOR WORD-TYPE ARRAY */
              R = INX(P);
              END;
            ELSE R = FINDAC;
            IF SFP = HALFWORD THEN TP = "48";
            ELSE TP = LOAD;
                CALL CHECK_COMMON;
         END;
```

```
                ELSE
                   DO;
                      R = FINDAC;
                      CALL EMITRR ("1B", R, R); /* CLEAR R */
                         TP = "43";
                         CALL CHECK_COMMON;
                      /* INSERT CHARACTER */
                      BASES(INX(P)) = AVAIL;
                   END;
                IF SFP = CHRTYPE THEN TYPE(P) = DESCRIPT;
                   ELSE TYPE(P) = ACCUMULATOR;
                REG(P) = R;
             END;
          END;
             ELSE IF TP = CONSTANT THEN
                DO;
                   R = FINDAC;
                   /* FETCH A CONSTANT INTO AN ACCUMULATOR */
                   IF FIXV(P) = 0 THEN CALL EMITRR("1B", R, R);
                   ELSE IF FIXV(P) < "1000" & FIXV(P) >= 1 THEN
                      CALL EMITRX(LA, R, 0, 0, FIXV(P));
                      ELSE
                         DO;
                            CALL EMITCONSTANT (FIXV(P));
                            CALL EMITRX (LOAD, R, 0, ADREG, ADRDISP);
                         END;
                   TYPE(P) = ACCUMULATOR;
                   REG(P) = R;
                END;
             ELSE IF TP = CHRTYPE THEN
                DO;
                   R = FINDAC;
                   TYPE(P) = DESCRIPT;
                   REG(P) = R;
                   T1 = VAR(P);
                   SD = LENGTH(T1) - 1;
                   IF SD < 0 THEN
                      CALL EMITRR("1B",R,R); /* CLEAR  REG R, NULL STRING */
                   ELSE
                      DO;
                         CALL FINDADDRESS (-DSP);
                         /* MAKE UP A DESCRIPTOR */
                         CALL EMITDESC(SHL(SD,24)+CHP);
                         DO I = 0 TO SD;
                            CALL EMITCHAR(BYTE(T1, I));
                         END;
                         CALL EMITRX (LOAD, R, 0, ADREG, ADRDISP);
                      END;
                END;
             ELSE IF TP ^= ACCUMULATOR THEN IF TP ^= DESCRIPT THEN
                CALL ERROR ('FORCEACCUMULATOR FAILED ***', 1);
   END FORCEACCUMULATOR;

FORCEDESCRIPT:
   PROCEDURE (P);
      /* GET A DESCRIPTOR FOR THE OPERAND P */
```

```
      DECLARE P FIXED;
      CALL FORCEACCUMULATOR (P);
      IF TYPE(P) ^= DESCRIPT THEN
         DO;
            CALL EMITRX (STORE, REG(P), 0, DBR, STRN);
            /* STORE IN PARAMETER LOCATION FOR NUMBER-TO -DECIMAL-STRING */
            BASES(REG(P)) = AVAIL;
            CALL CALLSUB(0,NMBRNTRY,3,P);
            /* ASSUMES NUMBER-TO-STRING IS IN THE 1ST PAGE */
            TYPE(P) = DESCRIPT;
         END;
   END FORCEDESCRIPT;

GENSTORE:
   PROCEDURE (MP, SP);
      DECLARE (MP, SP, SFP, SB, SD, I, TP) FIXED;
      COUNT#STORE = COUNT#STORE + 1;
      IF TYPE(SP) = SPECIAL THEN RETURN;
      /* GENERATE TYPE CONVERSION (IF NECESSARY) & STORAGE CODE --
            ALSO HANDLES OUTPUT AND FILE ON LEFT OF REPLACE OPERATOR */

      CHECK_COMMON:
         PROCEDURE;  /* STORE DATA INTO COMMON IF NECESSARY */
            DECLARE OFFSET FIXED;
            IF SYESDTYPE(I) ^= "05" THEN   /* NOT COMMON */
               DO;
                  IF (SYESDTYPE(FIXL(SP)) = "05" & MOVEFLAG) \
                        TYPE(SP) = DESCRIPT  THEN
                     DO;  /*MOVE THE STRING FROM COMMON OR ASSIGNMNET */
                        CALL EMITRX(STORE,SBR,0,DBR,STRREG); /* RECEIVE STR*/
                        CALL EMITRX(STORE,REG(SP),0,DBR,STRN); /*EXT DESC*/
                        BASES(REG(SP)) = AVAIL; /*FREE UP ACCUM */
                        CALL CALLSUB(0,STRMOVE,3,SP); /* MOVE THE STR*/
                        TYPE(SP) = DESCRIPT;
                     END;
                  CALL EMITRX(TP,REG(SP),INX(MP),SB,SD);
                  RETURN;
               END;
               CALL EMITRX(LOAD,BRCHREG,0,SB,SD); /* ADDR OF DATA */
               OFFSET = 0;
                     IF SFP = CHRTYPE  THEN /* INC PAST COM PREAMBLE*/
                        DO;
                           IF MOVEFLAG THEN
                              DO;
                                 CALL EMITRX(STORE,BRCHREG,0,DBR,STRREG);
                                  CALL EMITRX(STORE,REG(SP),0,DBR,STRN);
                                  BASES(REG(SP)) = AVAIL;
                                  CALL CALLSUB(0,STRMOVE,3,SP);
                                  TYPE(SP) = DESCRIPT;
                                   CALL EMITRX(LOAD,BRCHREG,0,DBR,STRREG);
                              END;
                        OFFSET = 20;
                        END;
            CALL EMITRX(TP,REG(SP),INX(MP),BRCHREG,OFFSET); /* STORE THE DATA */
      END CHECK_COMMON;
```

```
I  = FIXL(MP);
SB = SYBASE(I);
SD = SYDISP(I);
SFP = SYTYPE(I);
IF SFP = SPECIAL THEN
   DO;
      IF SD = 3 THEN      /*  FUNCTION BYTE ON THE LEFT */
         DO;
            CALL FORCEACCUMULATOR(SP);
            CALL EMITRX("42",REG(SP),INX(MP),REG(MP),0);
         END;
      ELSE IF SD = 7 THEN
         DO;      /* OUTPUT   */
            CALL EMITRX("90",1,3,DBR,BASEDATA+4);
            MOVEFLAG = FALSE;  /*TURN OFF MOVE FOR OUTPUT */
            TARGET_REGISTER = 0;
            CALL FORCEDESCRIPT (SP);
            TARGET_REGISTER = -1;
            IF REG(SP) ^= 0 THEN CALL EMITRR ("18", 0, REG(SP));
            IF REG(MP) = 0 THEN CALL EMITRR ("1B", 2, 2);
            ELSE IF REG(MP) ^= 2 THEN CALL EMITRR ("18", 2, REG(MP));
            BASES(REG(MP)) = AVAIL;
            CALL EMITRX(LA,SBR,0,DBR,SAVE); /* SAVE AREA ADDR */
            CALL EMITRX (LA, 1, 0, 0, 8);  /* 8 = PRINT CODE */
            CALL EMITRX(LOAD,CALLREG,0,DBR,XPLMON);
            CALL EMITRR (BALR, BRCHREG, CALLREG); /* MONITOR CALL */
            CALL EMITRX(LOAD,SBR,0,DBR,BASEDATA+52); /*RESTORE R13 */
            CALL EMITRX("98",1,3,DBR,BASEDATA+4);
            MOVEFLAG = TRUE;   /*RETURN FLAG TO NORMAL STATE */
         END;
      ELSE IF SD = 8 THEN
         CALL FILE_PSEUDO_ARRAY(SP,MP,4);
      ELSE CALL ERROR ('ILLEGAL USE OF ' \\ SYT(I));
   END;
ELSE
   DO;
      CALL FORCEACCUMULATOR (SP);
      IF TYPE(SP) ^= SPECIAL THEN
         DO;
            IF SFP=FIXEDTYPE & TYPE(SP)=ACCUMULATOR \ SFP=CHRTYPE THEN
               DO;
                  IF SFP = CHRTYPE THEN CALL FORCEDESCRIPT (SP);
                  /* SHIFT INDEX FOR WORD ARRAY */
                  IF INX(MP) ^= 0 THEN CALL EMITRX ("89", INX(MP),0,0,2);
                  TP = STORE; CALL CHECK_COMMON;
               END;
            ELSE IF SFP = HALFWORD & TYPE(SP) = ACCUMULATOR THEN
               DO;
                  IF INX(MP) ^= 0 THEN CALL EMITRR ("1A",INX(MP),INX(MP));
                  TP = "40"; CALL CHECK_COMMON;
               END;
            ELSE IF SFP = BYTETYPE & TYPE(SP) = ACCUMULATOR THEN
               DO;  TP = "42"; CALL CHECK_COMMON; END;
            ELSE CALL ERROR('ASSIGNMENT NEEDS ILLEGAL TYPE CONVERSION');
         END;
   END;
```

```
      BASES(INX(MP)) = AVAIL;
      BASES(REG(SP)) = AVAIL;
      CALL MOVESTACKS (SP, MP);
   END GENSTORE;

STRINGCOMPARE:
   PROCEDURE;
      /* GENERATES THE CODE TO COMPARE THE STRINGS AT SP & MP */
      DECLARE (I, J, K) FIXED;
      CALL FORCEDESCRIPT (SP);  /* GET THE DESCRIPTOR FOR THE SECOND OPERAND */
      I = 6 - REG(MP) - REG(SP);  /* FIND THE THIRD REGISTER */
      CALL EMITRR ("18", 0, REG(MP));  /* WE CAN USE 0 FOR SCRATCH */
      CALL EMITRR ("17", 0, REG(SP));     /* EXCL. \ TO COMPARE  */
      CALL EMITRX ("8A", 0, 0, 0, 24);  /* CHECK HIGH ORDER 8 BITS FOR ZERO */
      IF REG(MPP1) = 6 \ REG(MPP1) = 8 THEN
         DO;           /* IF WE ONLY NEED TO TEST EQUALITY, CODE IS SIMPLER  */
            K = PP;
            CALL BRANCH (6, 0);
         END;
      ELSE
         DO;
            J = PP;
            CALL BRANCH (8, 0);     /*  SKIP IF EQUAL LENGTH */
            CALL EMITRR ("15", REG(MP), REG(SP));     /* SET CONDITION CODE  */
            K = PP;  /* SAVE FOR FIXUP */
            CALL BRANCH ("F", 0);  /* BRANCH AROUND STRING COMPARE CODE  */
            CALL FIXBFW (J, PP);
         END;
      IF BASES(I) ~= AVAIL THEN CALL EMITRR ("18", 0, I);  /* SAVE REG I */
      CALL EMITRR ("18", I, REG(MP));
      CALL EMITRX ("88", I, 0, 0, 24);  /* SCALE LENGTH FOR EXECUTE COMMAND  */
      CALL EMITDATAWORD ("D5000000" + SHL(REG(MP), 12));
      CALL EMITBYTE (SHL(REG(SP), 4));
      CALL EMITBYTE (0);
      CALL FINDADDRESS (DP-6);
      CALL EMITRX ("44", I, 0, ADREG, ADRDISP);
      IF BASES(I) ~= AVAIL THEN CALL EMITRR ("18", I, 0);/* RESTORE REG I*/
      BASES(REG(SP)) = AVAIL;
      CALL FIXBFW (K, PP);  /* BRING OTHER BRANCH IN HERE */
   END STRINGCOMPARE;

SHOULDCOMMUTE:
   PROCEDURE BIT(1);
      IF TYPE(SP) = VARIABLE THEN
         IF SYTYPE(FIXL(SP)) = FIXEDTYPE THEN RETURN FALSE;
      IF TYPE(MP) = CONSTANT THEN RETURN TRUE;
      IF TYPE(MP) = VARIABLE THEN
         IF SYTYPE(FIXL(MP)) = FIXEDTYPE THEN RETURN TRUE;
      RETURN FALSE;
   END;

ARITHEMIT:
   PROCEDURE (OP);
      /* EMIT AN INSTRUCTION FOR AN INFIX OPERATOR -- CONNECTS MP & SP */

      DECLARE (OP, TP, T1) FIXED;
```

```
      COUNT#ARITH = COUNT#ARITH + 1;
      TP = 0;  /* REMEMBER IF COMMUTED */
      IF COMMUTATIVE(OP) THEN
            IF SHOULDCOMMUTE THEN
               DO;
                  TP = MP; MP = SP; SP = TP;
               END;
      CALL FORCEACCUMULATOR (MP);  /* GET THE LEFT ONE INTO AN ACCUMULATOR */
      /* FIXL(SP) IS GARBAGE IF TYPE ^= VARIABLE, WE GET OC5 IF WE TEST IT */
      T1 = "0";
      IF TYPE(SP) = VARIABLE THEN IF SYTYPE(FIXL(SP)) = FIXEDTYPE THEN T1 = "1";
      IF TYPE(MP) = DESCRIPT THEN
         DO;
            IF OP = CMPRR THEN CALL STRINGCOMPARE;
            ELSE CALL ERROR ('ARITHMETIC WITH A STRING DESCRIPTOR');
         END;
      ELSE IF T1 THEN
            DO;  /* OPERATE DIRECTLY FROM STORAGE  */
               IF INX(SP) ^= 0 THEN CALL EMITRX ("89", INX(SP), 0, 0, 2);
               /* SHIFT TO  WORD INDEXING */
               CALL EMITRX(OP+64,REG(MP),INX(SP),SYBASE(FIXL(SP)),
                  SYDISP(FIXL(SP)));
               /* REG OPCODE + 64 = RX OPCODE */
               BASES(INX(SP)) = AVAIL;
            END;
      ELSE IF TYPE(SP) = CONSTANT THEN
            DO;
               CALL EMITCONSTANT (FIXV(SP));
               CALL EMITRX (OP+64, REG(MP), 0, ADREG, ADRDISP);
            END;
      ELSE
            DO;
               CALL FORCEACCUMULATOR (SP);
               IF TYPE(SP) ^= ACCUMULATOR THEN
                  CALL ERROR ('ARITHMETIC BETWEEN STRING DESCRIPTORS', 1);
               CALL EMITRR (OP, REG(MP), REG(SP));
               BASES(REG(SP)) = AVAIL;
            END;
      IF TP ^= 0 THEN
         DO;  /* COMMUTED */
            SP = MP;  MP = TP;
            CALL MOVESTACKS (SP, MP);
         END;
         /* BY THE ALGORITHM, TYPE(MP) IS ALREADY ACCUMULATOR */
   END ARITHEMIT;

BOOLBRANCH:
   PROCEDURE (SP, MP);
      DECLARE (SP, MP, T1) FIXED;
      T1 = "0";
      IF TYPE(SP) = VARIABLE THEN IF SYTYPE(FIXL(SP)) = BYTETYPE THEN T1 = "1";
      /* GENERATE A CONDITIONAL BRANCH FOR A DO WHILE OR AN IF STATEMENT */
      IF STILLCOND ^= 0 THEN
         DO;
            BASES(REG(SP)) = AVAIL;
               IF PP < "1008" THEN PP = PP - 12; ELSE PP = PP - 16;
```

```
            IF CONTROL(BYTE(CON_E)) THEN
                  OUTPUT = X4 \\ '                    BACK UP CODE EMITTER';
            INSTRUCT(BC) = INSTRUCT(BC) - 1;  /* KEEP STATISTICS ACCURATE */
            INSTRUCT(LA) = INSTRUCT(LA) - 2;
            REG(SP) = STILLCOND;
         END;
      ELSE IF T1 THEN
         DO;
            IF INX(SP) ^= 0 THEN
               DO;
                  CALL EMITRR('1A",INX(SP),SYBASE(FIXL(SP)));
                  CALL EMITRX("91",0,1,INX(SP),SYDISP(FIXL(SP)));
                  /*  TEST UNDER MASK  */
                  BASES(INX(SP)) = AVAIL;
               END;
            ELSE CALL EMITRX("91",0,1,SYBASE(FIXL(SP)),SYDISP(FIXL(SP)));
                  /*  TEST UNDER MASK  */
            REG(SP) = 8;
         END;
      ELSE IF TYPE(SP) = CONSTANT THEN
         DO;
            IF FIXV(SP) THEN
               DO;  FIXL(MP) = 0;  RETURN; END;
            ELSE REG(SP) = 15;
         END;
      ELSE IF TYPE(SP) ^= CONDITION THEN
         DO;
            CALL FORCEACCUMULATOR (SP);
            CALL EMITRX ("54", REG(SP), 0, DBR, TRUELOC);/* TEST LS BIT */
            BASES(REG(SP)) = AVAIL;
            REG(SP) = 8;
         END;
      FIXL(MP) = PP;  /* SAVE ADDRESS FOR FUTURE FIXUP */
      CALL BRANCH (REG(SP), 0);  /* REG(SP) HAS THE CC TO BE TESTED FOR */
   END BOOLBRANCH;


SET_LIMIT:
   PROCEDURE;
      /*  SETS DO LOOP LIMIT FOR <ITERATION CONTROL>   */
      IF TYPE(MPP1) = CONSTANT THEN
         CALL EMITCONSTANT(FIXV(MPP1));
      ELSE
         DO;
            CALL FORCEACCUMULATOR(MPP1);
            CALL EMITDATAWORD(0);
            CALL FINDADDRESS(DP-4);
            CALL EMITRX(STORE,REG(MPP1),0,ADREG,ADRDISP);
            BASES(REG(MPP1)) = AVAIL;
         END;
      INX(MP) = ADREG;
      FIXV(MP) = ADRDISP;
   END  SET_LIMIT;


DIVIDE_CODE:
```

```
   PROCEDURE;
      /*  GENERATES THE CODE FOR DIVISION  */

      TARGET_REGISTER = 0;
      CALL FORCEACCUMULATOR(MP);
      TARGET_REGISTER = -1;
      IF REG(MP) ^= 0 THEN
         DO;
            CALL EMITRR("18",0,REG(MP));          /*  LR    0,REG(MP)  */
            BASES(REG(MP)) = AVAIL;
            REG(MP) = 0;
         END;
      IF BASES(1) = AVAIL THEN
         DO;
            /*  MUST "SMEAR"  THE SIGN  */
            CALL EMITRX("8E",0,0,0,32);           /*  SRDA  0,32       */
            BASES(1) = ACCUMULATOR;
            CALL ARITHEMIT("1D");                 /*  DIVIDE  */
            REG(MP) = 1;                          /*  RESULT  */
         END;
      ELSE
         CALL ERROR('DIVISION OR MOD REQUIRES BUSY REGISTER',1);

   END  DIVIDE_CODE;


SHIFT_CODE:
   PROCEDURE (OP);
      /*  GENERATES CODE FOR THE BUILT IN FUNCTIONS  SHL  AND  SHR  */
      DECLARE OP BIT (8);
      IF CNT(MP) ^= 2 THEN
         CALL ERROR('SHIFT REQUIRES TWO ARGUMENTS',0);
      ELSE IF TYPE(MPP1) = CONSTANT THEN
         DO;
            IF OP = "89" & FIXV(MPP1) = 1 THEN
               CALL EMITRR ("1A", REG(MP), REG(MP));
            ELSE CALL EMITRX (OP, REG(MP), 0, 0, FIXV(MPP1));
         END;
      ELSE
         DO;
            CALL FORCEACCUMULATOR(MPP1);
            CALL EMITRX(OP, REG(MP), 0, REG(MPP1), 0);
            BASES(REG(MPP1)) = AVAIL;
         END;
      TYPE(MP) = ACCUMULATOR;
   END  SHIFT_CODE;


  /*                    BUILT-IN FUNCTIONS                          */

CATENATE_CODE:
   PROCEDURE;
      /* BUILD A CATENATE SUBROUTINE  */
```

```
      CATENTRY = PP;
      CALL CHECK_STRING_OVERFLOW;
      CALL EMITRX (LOAD,1,0,A1,A2);  /*  LOAD FIRST DESCRIPTOR */
      CALL EMITRR ("18", 3, 1);  /* COPY INTO REG(3)  */
      CALL EMITRX ("5E", 3, 0, B1, B2);  /* COMBINE DESCRIPTORS  */
      CALL EMITRR ("12", 1, 1);  /* TEST FOR NULL FIRST OPERAND  */
      CALL EMITRR (BCR, 8, BRCHREG);  /* RETURN WITH RESULT IN REG(3)  */
      CALL EMITRR (CMPRR, 3, 1);  /* IS SECOND OPERAND NULL?  */
      CALL EMITRR (BCR, 8, BRCHREG);  /* RETURN WITH RESULT IN REG(3)  */
      CALL FINDADDRESS (MASKF000);
      CALL EMITRX ("54", 3, 0, ADREG, ADRDISP);  /* MASK OUT ADDRESS  */
      CALL EMITRX ("5E", 3, 0, DBR, CATCONST);  /* CORRECT LENGTH OF RESULT  */
      CALL FINDADDRESS (MOVER);  /* FIND MOVE INSTRUCTION */
      T1 = ADREG;  T2 = ADRDISP;
      CALL EMITRR ("18", 0, 3);  /* SAVE LENGTH IN REG(0)  */
      CALL EMITRX (LOAD,2,0,SBR,FREEPOINT);
      CALL EMITRX (CMPR, 1, 0, DBR, STRL); /* SKIP MOVE IF AT TOP */
      J = PP;
      CALL BRANCH (6, 0);        /* FAKE MOVE */
      CALL EMITRX (LA, 1, 0, 1, 0);
      CALL EMITRR ("16", 0, 1);
      K = PP;
      CALL BRANCH ("F", 0);
      CALL FIXBFW (J, PP);
      CALL EMITRR ("16",0,2);  /* OR IN CORRECT ADDRESS */
      CALL EMITRX ("43",3,0,A1,A2);  /* INSERT LENGTH FIELD */
      CALL EMITRX ("44",3,0,T1,T2);   /* EXECUTE THE MOVE */
      CALL EMITRX ("41",2,3,2,1);  /* UPDATE FREEPOINT */
      CALL FIXBFW (K, PP);
      CALL EMITRX (LOAD,1,0,B1,B2);  /* LOAD SECOND DESCRIPTOR */
      CALL EMITRX ("43", 3,0,B1,B2);  /* INSERT LENGTH FIELD  */
      CALL EMITRX ("44",3,0,T1,T2);  /* EXECUTE THE MOVE */
      CALL EMITRX ("41",2,3,2,1);   /*  UPDATE FREEPOINT  */
      CALL EMITRX (STORE, 0, 0, DBR, STRL);  /* STORE INTO STRL */
      CALL EMITRX (STORE,2,0,SBR,FREEPOINT);   /* SAVE TOP OF STR. A.  */
      CALL EMITRR ("18", 3, 0);  /* RESULT TO REG(3)  */
      CALL EMITRR (BCR, 15, BRCHREG);  /* RETURN  */
   END CATENATE_CODE;

CONVERT_CODE:
   PROCEDURE;
      /*  THE NUMBER-TO-STRING CONVERSION SUBROUTINE */

      NMBRNTRY = PP;
      CALL CHECK_STRING_OVERFLOW;          /* CALL COMPACTIFY */
      CALL EMITRX (LOAD, 3,0, DBR, STRN);
      CALL EMITRR ("10",3,3);              /* SET POSITIVE FOR CONVERT */
      CALL EMITRX (LOAD, 1,0,SBR,FREEPOINT);    /* FREE SOME STRING AREA */
      CALL EMITRX  (LA,1,0,1,11);          /* 11 IS THE MAXIMUM NUMBER OF DIGITS
                                              IN A CONVERTED 32 BIT INTEGER */
      CALL EMITRX (STORE,1,0,SBR,FREEPOINT);
      CALL EMITRX (LA,0,0,0,10);           /* BASE 10 FOR DIVISION */
      I = PP;
      CALL EMITRR ("06",1,0);              /* COUNT THE DIGIT */
      CALL EMITRR ("1B",2,2);              /* CLEAR REGISTER 2 */
      CALL EMITRR ("1D",2,0);              /* DIVIDE BY 10 */
```

```
      CALL EMITRX (LA,2,0,2,"F0");          /* ADD IN THE EBCDIC CODE */
      CALL EMITRX ("42",2,0,1,0);
      CALL EMITRR ("12",3,3);               /* TEST FOR ZERO */
      CALL BRANCH (6,I);                    /* GET NEXT DIGIT */
      CALL EMITRX (LOAD,3,0,DBR,STRN);
      CALL EMITRR ("12",3,3);               /* TEST FOR NEGATIVE */
      I = PP;
      CALL BRANCH (10,0);
      CALL EMITRX (LA,2,0,0,"60");          /* "60" = '-' */
      CALL EMITRR ("06",1,0);
      CALL EMITRX ("42",2,0,1,0);
      CALL FIXBFW (I,PP);
      CALL EMITRX(LOAD,3,0,SBR,FREEPOINT); /* MAKE UP RESULT DESCRIPTOR*/
      CALL EMITRR("1B",3,1);
      CALL EMITRR("06",3,0);
      CALL EMITRX("89",3,0,0,24);      /*SHIFT LENGTH FIELD LEFT  */
      CALL EMITRR("1A",3,1);           /* ADD IN ADDRESS */
      CALL EMITRX(STORE,3,0,DBR,STRL);  /* PTR TO NEW STRING */
      CALL EMITRR(BCR,15,BRCHREG);     /* RETURN */

   END CONVERT_CODE;

   STRING_MOVE:
     PROCEDURE;
      STRMOVE = PP;  /* ADDRESS OF ROUTINE */
      CALL EMITRX(LOAD,1,0,DBR,STRN);  /* GET DESCRIPTOR */
      CALL EMITRR("1B",3,3); /*CLEAR R3 FOR NEXT INSTR*/
      CALL EMITRR("12",1,1); /* CHECK FOR NULL */
      CALL EMITRR(BCR,8,BRCHREG); /* IF NULL RETURN 0 IN R3 */
      CALL EMITDATAWORD(0);
      CALL FINDADDRESS(DP-4);
      T1 = ADREG; T2 = ADRDISP;
      CALL EMITRX(STORE,SBR,0,T1,T2);  /* SAVE ORIGINAL SBR*/
      CALL EMITRX(LOAD,SBR,0,DBR,STRREG);/*GET BASE ADDR OF STRING CSECT*/
      CALL CHECK_STRING_OVERFLOW;
      CALL EMITRX(LOAD,2,0,SBR,FREEPOINT);
      CALL EMITRX("43",3,0,DBR,STRN);  /* GET LENGTH   */
      CALL FINDADDRESS(MOVER);
      CALL EMITRX("44",3,0,ADREG,ADRDISP); /* MOVE THE STRING */
      CALL EMITRR("18",1,3);   /* SAVE LENGTH */
      CALL EMITRX("89",3,0,0,24);   /* SHIFT LENGTH TO HIGH BYTE */
      CALL EMITRR("1A",3,2);        /* NEW DESCRIPTOR */
      CALL EMITRX("41",2,1,2,1);  /* UPDATE FREEPOINT */
      CALL EMITRX(STORE,3,0,DBR,STRL); /*LAST STRING */
      CALL EMITRX(STORE,2,0,SBR,FREEPOINT); /* STORE NEW VALUE */
      CALL EMITRX(LOAD,SBR,0,T1,T2); /* RESTORE */
      CALL EMITRR(BCR,15,BRCHREG);  /* RETURN */
      CALL FIXBFW (CATENTRY-4, PP);

   END STRING_MOVE;


  /*                    TIME AND DATE                              */

PRINT_TIME:
```

```
   PROCEDURE (MESSAGE, T);
      DECLARE MESSAGE CHARACTER, T FIXED;
      MESSAGE = MESSAGE \\ T/360000 \\ ':' \\ T MOD 360000 / 6000 \\ ':'
         \\ T MOD 6000 / 100 \\ PERIOD;
      T = T MOD 100;  /* DECIMAL FRACTION  */
      IF T < 10 THEN MESSAGE = MESSAGE \\ '0';
      OUTPUT = MESSAGE \\ T \\ PERIOD;
   END PRINT_TIME;

PRINT_DATE_AND_TIME:
   PROCEDURE (MESSAGE, D, T);
      DECLARE MESSAGE CHARACTER, (D, T, YEAR, DAY, M) FIXED;
      DECLARE MONTH(11) CHARACTER INITIAL ('JANUARY', 'FEBRUARY', 'MARCH',
         'APRIL', 'MAY', 'JUNE', 'JULY', 'AUGUST', 'SEPTEMBER', 'OCTOBER',
         'NOVEMBER', 'DECEMBER'),
      DAYS(12) FIXED INITIAL (0, 31, 60, 91, 121, 152, 182, 213, 244, 274,
         305, 335, 366);
      YEAR = D/1000 + 1900;
      DAY = D MOD 1000;
      IF (YEAR & "3") ^= 0 THEN IF DAY > 59 THEN DAY = DAY + 1; /* ^ LEAP YEAR*/
      M = 1;
      DO WHILE DAY > DAYS(M);  M = M + 1;  END;
      CALL PRINT_TIME(MESSAGE \\ MONTH(M-1) \\ X1 \\ DAY-DAYS(M-1) \\ COMMA
         \\ YEAR \\ ',  CLOCK TIME = ', T);
   END PRINT_DATE_AND_TIME;

  /*                          INITIALIZATION                                */


INITIALIZATION:
   PROCEDURE;
      EJECT_PAGE;
   CALL PRINT_DATE_AND_TIME ('X P L   COMPILATION -- WEST VIRGINIA UNIV -- XCOM
 IV VERSION OF ', DATE_OF_GENERATION, TIME_OF_GENERATION);
      DOUBLE_SPACE;
      CALL PRINT_DATE_AND_TIME ('TODAY IS ', DATE, TIME);
      DOUBLE_SPACE;
      DO I = 1 TO NT;
         S = V(I);
         IF S = '<NUMBER>' THEN NUMBER = I;  ELSE
         IF S = '<IDENTIFIER>' THEN IDENT = I;  ELSE
         IF S = '<STRING>' THEN STRING = I;  ELSE
         IF S = '/' THEN DIVIDE = I;  ELSE
         IF S = '_\_' THEN EOFILE = I;  ELSE
         IF S = 'DECLARE' THEN STOPIT(I) = TRUE;  ELSE
         IF S = 'PROCEDURE' THEN STOPIT(I) = TRUE;  ELSE
         IF S = 'END' THEN STOPIT(I) = TRUE;  ELSE
         IF S = 'DO' THEN STOPIT(I) = TRUE;  ELSE
         IF S = ';' THEN STOPIT(I) = TRUE;  ELSE
         IF S = OR THEN ORSYMBOL = I; ELSE
         IF S = '\\' THEN CONCATENATE = I; ELSE
         ;
      END;
      IF IDENT = NT THEN RESERVED_LIMIT = LENGTH(V(NT-1));
      ELSE RESERVED_LIMIT = LENGTH(V(NT));
```

```
V(EOFILE) = 'EOF';
STOPIT(EOFILE) = TRUE;
CHARTYPE(BYTE(X1)) = 1;
CHARTYPE(BYTE(HYPHEN)) = 2;
CHARTYPE(BYTE('*')) = 3;
DO I = 0 TO 255;
   NOT_LETTER_OR_DIGIT(I) = TRUE;
END;
DO I = 0 TO LENGTH(ALPHABET) - 1;
   J = BYTE(ALPHABET, I);
   TX(J) = I;
   NOT_LETTER_OR_DIGIT(J) = FALSE;
   CHARTYPE(J) = 4;
END;
DO I = 0 TO 9;
   J = BYTE('0123456789', I);
   NOT_LETTER_OR_DIGIT(J) = FALSE;
   CHARTYPE(J) = 5;
END;
DO I = V_INDEX(0) TO V_INDEX(1) - 1;
   J = BYTE(V(I));
   TX(J) = I;
   CHARTYPE(J) = 7;
END;
CHARTYPE(BYTE(OR)) = 8;
CHARTYPE(BYTE('/')) = 6;
COMMUTATIVE("14") = TRUE;
COMMUTATIVE("16") = TRUE;
COMMUTATIVE("1A") = TRUE;
RETURNED_TYPE = FIXEDTYPE;          /* DEFAULT RETURN TYPE */

LASTBASE = DBR;  BASES(LASTBASE) = 0;
/*              INITIALIZE SYMBOL TABLE VARIABLES */
PP = 0;    /*  CODE ORIGIN */
DP = 0;     /*  DATA ORIGIN  */
DSP = 20;
CHP = 1;
CODE_ESDID = 1;     /* FIRST ESDID FOR LINK EDITING */

PPLIM, DPLIM, CHPLIM = DISKBYTES;
   /* UPPER BOUND FOR EMITTER ARRAYS */
PPORG, DPORG, CHPORG = 0;
   /* LOWER BOUND FOR EMITTER ARRAYS */
CURCBLK, CURDBLK, CURSBLK = 0;
   /* CURRENT BLOCK OCCUPYING EMITTER ARRAYS */
SHORTCFIX, SHORTDFIX, LONGCFIX, LONGDFIX = 0;
   /* STATISTICAL COUNTERS FOR FIXUPS */
FCP = 0;  /* POINTER INTO FIXUP ARRAY */
NDECSY ,PROCMARK = 1;   PARCT = 0;
/* INTEGERS FOR BRANCH ADDRESSING */
DO I = 0 TO PROGRAMSIZE;  CALL EMITDATAWORD(SHL(I,12)); END;

/*  WARNING, THE FOLLOWING SECTION OF INITIALIZE DEPENDS ON
    THE INITIALIZATION OF THE BUILTIN FUNCTION AND PSEUDO
    VARIABLE NAMES AND ATTRIBUTES IN THE SYMBOL TABLE ARRAYS.
*/
```

```
    SYDISP(2) = DP;                          /*  MONITOR_LINK                */
    BASEDATA = DP;   /* ADDRESS OF REG INITIALIZATION AREA */
    DP = DP + 64;    /* REG INIT AREA I.E.  16 * 4          */
    XPLCOMP = BASEDATA + 56;  /* ADDRESS OF COMPACTIFY */
    XPLMON = BASEDATA + 60;  /* RELOCATED ADDRESS OF THE MONITOR */
    SAVE = DP;       /* LINK SAVE AREA                      */
    DP = DP + 72;    /* 18 WORDS                            */

    MASKF000 = DP;  CALL EMITDATAWORD("FF000000");

    /*  SET UP THE MOVE TEMPLATE IN DATA AREA */

    MOVER = DP;
    CALL EMITBYTE("D2");  /* MVC */
    CALL EMITBYTE(0);
    CALL EMITBYTE("20");
    CALL EMITBYTE(0);
    CALL EMITBYTE("10");
    CALL EMITBYTE(0);
    SYDISP(3) = DP;                          /*  TIME_OF_GENERATION          */
    CALL EMITDATAWORD(TIME);
    SYDISP(4) =  DP;                         /*  DATE_OF_GENERATION          */
    CALL EMITDATAWORD(DATE);
    SYDISP(5) = 0;                           /*  COREWORD                    */
    SYDISP(6) = 0;                           /*  COREBYTE                    */
    SYDISP(9) = DP;                          /*  NDESCRIPT                   */
    DESCL = DP;
    CALL EMITDATAWORD (0);
    A1, B1 = SBR;  /* A1,A2 IS THE FIRST PARAMETER TO \\, */
    A2 = DSP;       /* B1,B2 IS THE SECOND */
    CALL EMITDESC (0);
    B2 = DSP;
    CALL EMITDESC (0);
    STRL = DP; CALL EMITDATAWORD(0);
    STRN = DP; CALL EMITDATAWORD(0);
    STRREG=DP; CALL EMITDATAWORD(0);
    TRUELOC = DP; CALL EMITDATAWORD(TRUE);
    COMPLOC = DP; CALL EMITDATAWORD("FFFFFFFF");
    CATCONST = DP; CALL EMITDATAWORD ("1000000");
    RTNADR = DP; CALL EMITDATAWORD (0);
    NDECSY = 30; PROCMARK=NDECSY+1;   /* POINTER TO XPLMAIN  */
    CURRENT_PROCEDURE = X1 \\ 'XPLMAIN';
    CALL EMITDATAWORD (0);
    SYDISP(NDECSY-2) = DP-4;                 /*  COMPACTIFY                  */
    STRING_RECOVER = NDECSY-2;
    SYESDTYPE(NDECSY-2) = "02";  /* ENTRY FOR XPLPACK OR COMPACTIFY */
    SYRLDADDR(NDECSY-2) = BASEDATA + 56;
    SYESDTYPE(NDECSY-1) = "02"; /* ENRTY FOR XPLMON */
    SYRLDADDR(NDECSY-1) = BASEDATA + 60;

/* SET UP MAIN ENTRY AND INITIALIZE REGISTERS  */

    SYESDTYPE(NDECSY) = "01";   /* USED FOR RELOCATING MAIN ENTRY */
    CALL BUILD_PROLOGUE(NDECSY);
    MAINLOC = SYDISP(NDECSY);
```

```
      CALL BRANCH('F',0);          /* BRANCH AROUND BUILT-INS */

      CALL CLEARREGS;

      /*          EMIT CODE FOR BUILT_IN FUNCTIONS                          */

      CALL CATENATE_CODE;

      CALL CONVERT_CODE;

      CALL STRING_MOVE;

      CODE_ESDID = 1;
      CALL CLEARREGS;

   /* ENABLE CONTROL TOGGLES FOR LISTING & SYMBOL DUMP  */
      CONTROL(BYTE('L')) = TRUE;
      CONTROL(BYTE('D')) = TRUE;

      /* FIRST SET UP GLOBAL VARIABLES CONTROLLING SCAN, THEN CALL IT */
      CP = 0;  TEXT_LIMIT = -1;
      TEXT, CURRENT_PROCEDURE = X0;
      CALL SCAN;

      /* INITIALIZE THE PARSE STACK */
      SP = 1;  PARSE_STACK(SP) = EOFILE;

   END INITIALIZATION;


  /*                SYMBOL AND STATISTICS PRINTOUT                          */


SYMBOLDUMP:
   PROCEDURE;
      /* LISTS THE SYMBOLS IN THE PROCEDURE THAT HAS JUST BEEN
         COMPILED IF $S OR $D IS ENABLED
         MAINTAIN PARITY ON $D AND $S
      */
      DECLARE (LPM, I, J, K, L, M) FIXED;
      DECLARE (BUFFER, BLANKS) CHARACTER;
      DECLARE EXCHANGES BIT(1), SYTSORT(SYTSIZE) BIT(16);

      OUTLINE:
         PROCEDURE (NAME, P) CHARACTER;
            DECLARE NAME CHARACTER, (P, B, D) FIXED;
            IF SYTYPE(P) = LABELTYPE \ SYTYPE(P) = CHAR_PROC_TYPE THEN
               DO;
                  B = PBR;
                  D = SHL(SYBASE(P), 12) + SYDISP(P);
               END;
            ELSE
               DO;
                  B = SYBASE(P);
```

```
            D = SYDISP(P);
         END;

      BUFFER = PAD (D \\ LPAREN \\ B \\ '),', 11);
      RETURN NAME \\ ': ' \\ TYPENAME(SYTYPE(P)) \\ ' AT ' \\ BUFFER \\
         ' DECLARED ON LINE ' \\ DECLARED_ON_LINE(P) \\
         ' AND REFERENCED ' \\ SYTCO(P) \\ ' TIMES.';
   END  OUTLINE;


IF PROCMARK <= NDECSY THEN
   DO;
      DOUBLE_SPACE;
      OUTPUT = 'SYMBOL  TABLE  DUMP';
      DOUBLE_SPACE;
      LPM = LENGTH(SYT(PROCMARK));
      L = 15;
      DO I = PROCMARK TO NDECSY;
         IF LENGTH(SYT(I)) > L THEN
            L = LENGTH(SYT(I));
      END;
      IF L > 70 THEN L = 70;
      BLANKS = SUBSTR(X70, 0, L);
      DO I = PROCMARK TO NDECSY;
         SYTSORT(I) = I;
         K = LENGTH(SYT(I));
         IF K > 0 THEN
            IF K < L THEN
               DO;
                  BUFFER = SUBSTR(BLANKS,K);
                  SYT(I) = SYT(I) \\ BUFFER;
               END;
            ELSE
               DO;
                  BUFFER = SUBSTR (SYT(I), 0, L);
                  SYT(I) = BUFFER;
               END;
      END;

      EXCHANGES = TRUE;
      K = NDECSY - PROCMARK;

      DO WHILE EXCHANGES;
         EXCHANGES = FALSE;
         DO J = 0 TO K - 1;
            I = NDECSY - J;
            L = I - 1;
            IF SYT(SYTSORT(L)) > SYT(SYTSORT(I)) THEN
               DO;
                  M = SYTSORT(I);
                  SYTSORT(I) = SYTSORT(L);
                  SYTSORT(L) = M;
                  EXCHANGES = TRUE;
                  K = J;          /* RECORD LAST SWAP */
               END;
         END;
```

```
            END;

            I = PROCMARK;
            DO WHILE LENGTH(SYT(SYTSORT(I))) = 0;
               I = I + 1;                  /* IGNORE NULL NAMES */
            END;

            DO I = I TO NDECSY;
               K = SYTSORT(I);
               OUTPUT = OUTLINE(SYT(K), K);

               K = K + 1;
               DO WHILE (LENGTH(SYT(K)) = 0) & (K <= NDECSY);
                  J = K - SYTSORT(I);
                  OUTPUT =
                     OUTLINE('  PARAMETER  ' \\ J \\ SUBSTR(BLANKS, 14), K);
                  K = K + 1;
               END;

            END;

            BUFFER = SUBSTR(SYT(PROCMARK), 0 , LPM);
            SYT(PROCMARK) = BUFFER;
            EJECT_PAGE;
         END;

   END  SYMBOLDUMP;


DUMPIT:
   PROCEDURE;     /* DUMP OUT THE COMPILED CODE & DATA AREAS  */
   DECLARE DUMPMSG CHARACTER INITIAL (                                    '
 MACRO DEFINITIONS:      LITERALLY:      IDCOMPARES          = SYMBOL TABLE SIZE =
 MACRO DEFINITIONS = STACKING DECISIONS= SCAN                = EMITRR            =
 EMITRX            = FORCEACCUMULATOR  = ARITHEMIT          = GENSTORE          =
');
   DECLARE DUMPMSG1 CHARACTER INITIAL (                                   '
 FIXBFW            = FIXDATAWORD       = FIXCHW             = GETDATA           =
 GETCODE           = FINDADDRESS       = SHORTCFIX          = LONGCFIX          =
 SHORTDFIX         = LONGDFIX          = INSTRUCTION FREQUENCIES:');
      CALL SYMBOLDUMP;

      IF ~ CONTROL(BYTE('L')) THEN RETURN; /* STOP LIST HERE */

      DOUBLE_SPACE;
      OUTPUT = SUBSTR(DUMPMSG,1,21);
      DOUBLE_SPACE;
      DO I = 0 TO TOP_MACRO;
      OUTPUT = FAD(MACRO_NAME(I), 21) \\ SUBSTR(DUMPMSG,21,21) \\ MACRO_TEXT(I);
      END;
      DOUBLE_SPACE;
      /*  PUT OUT THE ENTRY COUNT FOR IMPORTANT PROCEDURES */

      OUTPUT = SUBSTR(DUMPMSG,41,21) \\ IDCOMPARES;
      OUTPUT = SUBSTR(DUMPMSG,61,21) \\ MAXNDECSY;
```

```
      OUTPUT = SUBSTR(DUMPMSG,81,21) \\ TOP_MACRO + 1;
      OUTPUT = SUBSTR(DUMPMSG,101,21) \\ COUNT#STACK;
      OUTPUT = SUBSTR(DUMPMSG,121,21) \\ COUNT#SCAN;
      OUTPUT = SUBSTR(DUMPMSG,141,21) \\ COUNT#RR;
      OUTPUT = SUBSTR(DUMPMSG,161,21) \\ COUNT#RX;
      OUTPUT = SUBSTR(DUMPMSG,181,21) \\ COUNT#FORCE;
      OUTPUT = SUBSTR(DUMPMSG,201,21) \\ COUNT#ARITH;
      OUTPUT = SUBSTR(DUMPMSG,221,21) \\ COUNT#STORE;
      OUTPUT = SUBSTR(DUMPMSG1,1,21) \\ COUNT#FIXBFW;
      OUTPUT = SUBSTR(DUMPMSG1,21,21) \\ COUNT#FIXD;
      OUTPUT = SUBSTR(DUMPMSG1,41,21) \\ COUNT#FIXCHW;
      OUTPUT = SUBSTR(DUMPMSG1,61,21) \\ COUNT#GETD;
      OUTPUT = SUBSTR(DUMPMSG1,81,21) \\ COUNT#GETC;
      OUTPUT = SUBSTR(DUMPMSG1,101,21) \\ COUNT#FIND;
      OUTPUT = SUBSTR(DUMPMSG1,121,21) \\ SHORTCFIX;
      OUTPUT = SUBSTR(DUMPMSG1,141,21) \\ LONGCFIX;
      OUTPUT = SUBSTR(DUMPMSG1,161,21) \\ SHORTDFIX;
      OUTPUT = SUBSTR(DUMPMSG1,181,21) \\ LONGDFIX;
       DOUBLE_SPACE;
      OUTPUT = SUBSTR(DUMPMSG1,201,25);
       DOUBLE_SPACE;
      OUTPUT = X0;
      DO I = 0 TO 255;
         IF INSTRUCT(I) ^ = 0 THEN
             OUTPUT = SUBSTR(OPNAMES,OPER(I),4) \\ X4 \\ INSTRUCT(I);
      END;
      EJECT_PAGE;
   END DUMPIT;


STACK_DUMP:
   PROCEDURE;
      DECLARE LINE CHARACTER;
      LINE = 'PARTIAL PARSE TO THIS POINT IS: ';
      DO I = 2 TO SP;
         IF LENGTH(LINE) > 105 THEN
            DO;
               OUTPUT = LINE;
               LINE = X4;
            END;
         LINE = LINE \\ X1 \\ V(PARSE_STACK(I));
      END;
      OUTPUT = LINE;
   END STACK_DUMP;


  /*                  THE SYNTHESIS ALGORITHM FOR XPL                  */


SYNTHESIZE:
PROCEDURE(PRODUCTION_NUMBER);
   DECLARE PRODUCTION_NUMBER FIXED;

   /*  ONE STATEMENT FOR EACH PRODUCTION OF THE GRAMMAR*/
```

```
DO CASE PRODUCTION_NUMBER;
   ;      /*  CASE 0 IS A DUMMY, BECAUSE WE NUMBER PRODUCTIONS FROM 1  */

 /*  <PROGRAM>  ::=  <STATEMENT LIST>    */
   DO;   /* FINAL CODE FOR XPLSM INTERFACE & SETUP  */
      IF MP ^= 2 THEN  /* WE DIDN'T GET HERE LEGITIMATELY  */
         DO;
            CALL ERROR ('EOF AT INVALID POINT', 1);
            CALL STACK_DUMP;
         END;
      DO I = 1 TO NDECSY;
         IF SYTYPE(I) = FORWARDTYPE \ SYTYPE(I) = FORWARDCALL THEN
          IF SYTCO(I) > 0 THEN
            CALL ERROR ('UNDEFINED LABEL OR PROCEDURE: ' \\ SYT(I), 1);
      END;
      CALL EMITRR ("1B", 3, 3);  /* RETURN CODE OF ZERO  */
      CALL EMIT_RETURN;
      DO I = 4 TO DBR-1;
         CALL FIXWHOLEDATAWORD(BASEDATA+SHL(I,2), BASES(I));
      END;

  /* INSERT PROLOGUE FOR STRINGS */
   DESC(0) = 0;      /* LOWER_BOUND FOR XPLPACK  */
   DESC(1),DESC(2) = DSP + CHP; /* FREEPOINT AND FREEBASE */
   DESC(3) = DSP + SHL(DSP,5) + SHL(DSP,4); /* ABSOLUTE STRING TOP */

 /* ADD OFFSET OF DSP TO THE DESCRIPORS FOR PARTIAL RELOCATION */
   DO I = 5 TO SHR(DSP,2);
      IF DESC(I) ^= 0 THEN   /* ADD TOP OF DESCRIPTOR ADDR TO DESC  */
          DO;
             J = ((DESC(I) & "FF000000") + DSP);
             DESC(I) =  J + (DESC(I) & "00FFFFFF");
          END;
   END;

   DESC(4) = (SHR(DSP,2)); /* # OF DESCRIPTORS INCLUDING 5 FOR PREFACE*/

      COMPILING = FALSE;
   END;

 /*  <STATEMENT LIST> ::= <STATEMENT>    */
   ;
 /*  <STATEMENT LIST> ::= <STATEMENT LIST> <STATEMENT>    */
   ;
 /*  <STATEMENT> ::= <BASIC STATEMENT>    */
   DO;
      CALL CLEARREGS;
      STATEMENT_COUNT = STATEMENT_COUNT + 1;
      ESDTYPE  = "0";  /* RESET EXTERNAL/PUBLIC BIT */
   END;

 /*  <STATEMENT> ::= <IF STATEMENT>    */
   CALL CLEARREGS;

 /*  <BASIC STATEMENT> ::= <ASSIGNMENT> ;    */
   ;
```

```
/*  <BASIC STATEMENT> ::= <GROUP> ;     */
  ;
/*  <BASIC STATEMENT> ::= <PROCEDURE DEFINITION> ;     */
  ;
/*  <BASIC STATEMENT> ::= <RETURN STATEMENT> ;     */
  ;
/*  <BASIC STATEMENT> ::= <CALL STATEMENT> ;     */
  ;
/*  <BASIC STATEMENT> ::= <GO TO STATEMENT> ;     */
  ;
/*  <BASIC STATEMENT> ::= <DECLARATION STATEMENT> ;     */
  ;

/*  <BASIC STATEMENT> ::= ;     */
    ;
/*  <BASIC STATEMENT> ::= <LABEL DEFINITION> <BASIC STATEMENT>     */
    ;
/*  <IF STATEMENT> ::= <IF CLAUSE> <STATEMENT>     */
  CALL FIXBFW(FIXL(MP), PP); /* FIX THE ESCAPE BRANCH NOW THAT STMT IS DONE */

/*  <IF STATEMENT> ::= <IF CLAUSE> <TRUE PART> <STATEMENT>     */
  DO;  /* THERE ARE TWO BRANCHES TO BE FILLED IN WITH ADDRESSES HERE */
     CALL FIXBFW(FIXL(MPP1), PP); /* ESCAPE FROM TRUE PART */
     CALL FIXBFW(FIXL(MP), FIXV(MPP1)); /* HOP AROUND TRUE PART */
  END;

/*  <IF STATEMENT> ::= <LABEL DEFINITION> <IF STATEMENT>     */
  ;

/*  <IF CLAUSE> ::= IF <EXPRESSION> THEN     */
  CALL BOOLBRANCH(MPP1, MP); /* BRANCH ON FALSE OVER TRUE PART */

 /*  <TRUE PART> ::= <BASIC STATEMENT> ELSE     */
  DO;  /* SAVE THE PROGRAM POINTER & EMIT THE CONDITIONAL BRANCH */
     FIXL(MP) = PP;
     CALL BRANCH("F", 0); /* "F" MEANS UNCONDITIONAL BRANCH */
     FIXV(MP) = PP;
  END;

/*  <GROUP> ::= <GROUP HEAD> <ENDING>     */
  DO;  /* BRANCH BACK TO LOOP & FIX ESCAPE JUMP */
     IF INX(MP) = 1 \ INX(MP) = 2 THEN
        DO; /* STEP OR WHILE LOOP FIX UP */
           CALL BRANCH("F", PPSAVE(MP));
           CALL FIXBFW(FIXL(MP), PP);
        END;
     ELSE IF  INX(MP) = 3 THEN
        DO;  /* COMMENT  CASE GROUP */
           /* JUSTIFY TO WORD BOUNDARY */
           DP = (DP + 3) & "FFFFFC";
           CALL FINDADDRESS(DP);
           CALL FIXCHW(FIXL(MP)+2, SHL(ADREG,4)+SHR(ADRDISP,8), ADRDISP);
           DO I = PPSAVE(MP) TO CASEP-1; CALL EMITDATAWORD(CASESTACK(I)); END;
           CASEP = PPSAVE(MP) - 1;
           CALL FIXBFW(FIXV(MP), PP);
        END;
```

```
      IF LENGTH(VAR(SP)) > 0 THEN IF VAR(MP-1) ~= VAR(SP) THEN
         CALL ERROR ('END ' \\ VAR(SP) \\ '  MUST MATCH LABEL ON GROUP', 0);
   END;

/*  <GROUP HEAD> ::= DO ;     */
   INX(MP) = 0;

/*  <GROUP HEAD> ::= DO <STEP DEFINITION> ;     */
   DO;
      CALL MOVESTACKS(MPP1, MP);
      INX(MP) = 1;  /* 1 DENOTES STEP */
   END;

/*  <GROUP HEAD> ::= DO <WHILE CLAUSE> ;     */
   DO;
      PPSAVE(MP) = PPSAVE(MPP1);
      FIXL(MP) = FIXL(MPP1);
      INX(MP) = 2;  /* 2 DENOTES WHILE */
   END;

/*  <GROUP HEAD> ::= DO <CASE SELECTOR> ;     */
   DO;
      CALL MOVESTACKS(MPP1, MP);
      INX(MP) = 3;  /* 3 DENOTES CASE  */
      INFORMATION = INFORMATION \\ ' CASE 0.';
   END;

/*  <GROUP HEAD> ::= <GROUP HEAD> <STATEMENT>     */
   IF  INX(MP) = 3 THEN
      DO;  /* CASE GROUP, MUST RECORD STATEMENT ADDRESSES */
         CALL BRANCH ("F", FIXV(MP));
      IF CASEP >= CASELIMIT THEN CALL ERROR ('TOO MANY CASES', 1);
         ELSE CASEP = CASEP + 1;  CASESTACK(CASEP) = PP;
      IF BCD ~= 'END' THEN
            INFORMATION = INFORMATION \\ ' CASE ' \\ CASEP-PPSAVE(MP) \\ PERIOD;
      END;

/*  <STEP DEFINITION> ::= <VARIABLE> <REPLACE> <EXPRESSION> <ITERATION CONTROL>
      */
   DO; /* EMIT CODE FOR STEPPING DO LOOPS */
      CALL FORCEACCUMULATOR(MP+2);
      IF SYESDTYPE(FIXL(MP)) ~= "00" THEN
         CALL ERROR('EXTERNAL USED AS DO LOOP VARIABLE',1);
      IF INX(MP) ~ = 0 THEN
         CALL ERROR ('SUBSCRIPTED DO VARIABLE', 0);
      STEPK = PP;
      CALL BRANCH("F", 0);
      PPSAVE(MP) = PP;
      L = FIXL(MP);
      ADREG = SYBASE(L);
      ADRDISP = SYDISP(L);
      IF SYTYPE(L) = BYTETYPE THEN
         DO;
            CALL EMITRR("1B", REG(MP+2), REG(MP+2));
            CALL EMITRX ("43", REG(MP+2), 0, ADREG, ADRDISP);
         END;
```

```
      ELSE IF SYTYPE(L) = HALFWORD THEN
         CALL EMITRX ("48", REG(MP+2), 0, ADREG, ADRDISP);
      ELSE
         CALL EMITRX (LOAD, REG(MP+2), 0, ADREG, ADRDISP);
      CALL EMITRX ("5A", REG(MP+2), 0, REG(MP+3), FIXL(MP+3));
      CALL FIXBFW(STEPK, PP);
      IF SYTYPE(L) = BYTETYPE THEN I = "42";
      ELSE IF SYTYPE(L) = HALFWORD THEN I = "40";
      ELSE I = STORE;
      CALL EMITRX (I, REG(MP+2), 0, ADREG, ADRDISP);
      CALL EMITRX (CMPR, REG(MP+2), 0, INX(MP+3), FIXV(MP+3));
      FIXL(MP) = PP;
      CALL BRANCH("2", 0);
      BASES(INX(MP)) = AVAIL;
      BASES(REG(MP+2)) = AVAIL;
   END;

/*  <ITERATION CONTROL> ::= TO <EXPRESSION>     */
   DO;
      REG(MP) = DBR;
      FIXL(MP) = TRUELOC;  /* POINT AT THE CONSTANT ONE FOR STEP  */
      CALL SET_LIMIT;
   END;

/*  <ITERATION CONTROL> ::= TO <EXPRESSION> BY <EXPRESSION>     */
   DO;
      IF TYPE(SP) = CONSTANT THEN CALL EMITCONSTANT (FIXV(SP));
      ELSE
         DO;
            CALL FORCEACCUMULATOR (SP);
            CALL EMITDATAWORD (0);
            CALL FINDADDRESS (DP-4);
            CALL EMITRX (STORE, REG(SP), 0, ADREG, ADRDISP);
            BASES(REG(SP)) = AVAIL;
         END;
      REG(MP) = ADREG;
      FIXL(MP) = ADRDISP;
      CALL SET_LIMIT;
   END;

/*  <WHILE CLAUSE> ::= WHILE <EXPRESSION>     */
   CALL BOOLBRANCH(SP, MP);

/*  <CASE SELECTOR> ::= CASE <EXPRESSION>     */
   DO;
      CALL FORCEACCUMULATOR(SP);
      CALL EMITRX("89", REG(SP), 0, 0, 2);
      FIXL(MP) = PP;
      CALL EMITRX(LOAD, REG(SP), REG(SP), 0, 0);
      CALL EMITRX(BC, "F", REG(SP), PBR, 0);
      BASES(REG(SP)) = AVAIL;
      FIXV(MP) = PP;
      CALL BRANCH("F", 0);
      IF CASEP >= CASELIMIT THEN CALL ERROR ('TOO MANY CASES', 1);
      ELSE CASEP = CASEP + 1;
      CASESTACK(CASEP) = PP;
```

```
      PPSAVE(MP) = CASEP;
   END;

/*  <PROCEDURE DEFINITION> ::= <PROCEDURE HEAD> <STATEMENT LIST> <ENDING>    */
   DO; /* PROCEDURE IS DEFINED, RESTORE SYMBOL TABLE */
      IF LENGTH(VAR(SP)) > 0 THEN
         IF SUBSTR(CURRENT_PROCEDURE, 1) ^= VAR(SP) THEN
            CALL ERROR ('PROCEDURE' \\ CURRENT_PROCEDURE \\ ' CLOSED BY END ' \\
               VAR(SP), 0);
      IF CONTROL(BYTE('S')) THEN
         CALL SYMBOLDUMP;
      DO I = PROCMARK TO NDECSY;
         IF SYTYPE(I) = FORWARDTYPE \ SYTYPE(I) = FORWARDCALL THEN
          IF SYTCO(I) > 0 THEN
            CALL ERROR ('UNDEFINED LABEL OR PROCEDURE: ' \\ SYT(I), 1);
         IF SYESDTYPE(I) = "05" THEN    /* EXTERNAL USED AS A FORMAL PARM */
            CALL ERROR('COMMON VARIABLE DEFINED WITHIN A PROCEDURE',1);
      END;
      DO I = PROCMARK + PARCT  TO  NDECSY + 1;
         SYT(I) = X1;
      END;
      NDECSY = PROCMARK + PARCT - 1;
      /* PARAMETER ADDRESS MUST BE SAVED BUT NAMES DISCARDED */
      DO I = PROCMARK TO NDECSY;
         IF SYTYPE(I) = 0 THEN
            DO;
               CALL ERROR('UNDECLARED PARAMETER:' \\ SYT(I));
               SYTYPE(I) = FIXEDTYPE;
               CALL EMITDATAWORD(0);
               CALL FINDADDRESS(DP-4);
               SYBASE(I) = ADREG;
               SYDISP(I) = ADRDISP;
            END;
         SYT(I) = X0;
      END;
      /* EMIT A GRATUITOUS RETURN */
      RTNADR = PPSAVE(MP);
      CALL EMIT_RETURN;
      CURRENT_PROCEDURE = VAR(MP);
      PROCMARK = FIXV(MP);  PARCT = CNT(MP);
      RETURNED_TYPE = TYPE(MP) ;
      CALL FIXBFW(FIXL(MP), PP); /* COMPLETE JUMP AROUND PROCEDURE DEFINITION */
   END;

/*  <PROCEDURE HEAD> ::= <PROCEDURE NAME> ;     */
   DO;  /* MUST POINT AT FIRST PARAMETER EVEN IF NONEXISTENT  */
      /* SAVE OLD PARAMETER COUNT */
      CNT(MP) = PARCT; PARCT = 0;
      /* SAVE OLD PROCEDURE MARK IN SYMBOL TABLE */
      FIXV(MP) = PROCMARK;  PROCMARK = NDECSY + 1;
      TYPE(MP) = RETURNED_TYPE;
      RETURNED_TYPE = 0;
      CALL PROC_START;
   END;
```

```
/*  <PROCEDURE HEAD> ::= <PROCEDURE NAME> <TYPE> ;     */
  DO;
     CNT(MP) = PARCT;
     PARCT = 0;
     FIXV(MP) = PROCMARK;
     PROCMARK = NDECSY + 1;
     TYPE(MP) = RETURNED_TYPE;
     RETURNED_TYPE = TYPE(SP-1);
     IF RETURNED_TYPE = CHRTYPE THEN
        SYTYPE(FIXL(MP)) = CHAR_PROC_TYPE ;
     CALL PROC_START;
  END;

/*  <PROCEDURE HEAD> ::= <PROCEDURE NAME> <PARAMETER LIST> ;     */
  DO;
     CNT(MP) = CNT(MPP1);  /* SAVE PARAMETER COUNT */
     FIXV(MP) = FIXV(MPP1);
     TYPE(MP) = RETURNED_TYPE;
     RETURNED_TYPE = 0;
     CALL PROC_START;
  END;


/*  <PROCEDURE HEAD> ::= <PROCEDURE NAME> <PARAMETER LIST> <TYPE> ;        */
  DO;
     CNT(MP) = CNT(MPP1);
     FIXV(MP) = FIXV(MPP1);
     TYPE(MP) = RETURNED_TYPE;
     RETURNED_TYPE = TYPE(SP-1);
     IF RETURNED_TYPE = CHRTYPE THEN
        SYTYPE(FIXL(MP)) = CHAR_PROC_TYPE ;
     CALL PROC_START;
  END;

/*  <PROCEDURE NAME> ::= <LABEL DEFINITION> PROCEDURE     */
  DO;
     S = CURRENT_PROCEDURE;
     CURRENT_PROCEDURE = X1 \\ VAR(MP);
     VAR(MP) = S;
  END;

/*  <PARAMETER LIST> ::= <PARAMETER HEAD> <IDENTIFIER> )     */
  DO;
     PARCT = PARCT + 1;
     CALL ENTER (VAR(MPP1), 0, 0, 0);
     IF ESDTYPE = '02' THEN   /* THIS IS AN EXTERNAL PARAMETER */
        DO;
           CALL EMITDATAWORD(0);  /*VALUE OF PARM */
           CALL FINDADDRESS(DP-4);
           SYBASE(NDECSY) = ADREG;
           SYDISP(NDECSY) = ADRDISP;
           SYESDTYPE(NDECSY) = 'OF';  /* 'OF' = EXTERNAL PARAMETER */
        END;
     ELSE
     IF ESDTYPE = '01' THEN    /* PUBLIC PROC PARM */
        DO;
```

```
            SYTCO(NDECSY) = PARCT;  /* LOCATION OF PARM IN LIST */
            SYESDTYPE(NDECSY) = "0E";  /* USED LATER IN DECLARE GRAMMAR*/
         END;

   END;

/*  <PARAMETER HEAD> ::= (    */
   DO;  /* POINT AT THE FIRST PARAMETER FOR SYMBOL TABLE */
      FIXV(MP) = PROCMARK;  PROCMARK = NDECSY + 1;
      CNT(MP) = PARCT;
      PARCT = 0;
   END;

/*  <PARAMETER HEAD> ::= <PARAMETER HEAD> <IDENTIFIER> ,     */
   DO;
      PARCT = PARCT + 1;
      CALL ENTER (VAR(MPP1), 0, 0, 0);
      IF ESDTYPE = "02" THEN   /* THIS IS AN EXTERNAL PARAMETER */
         DO;
            CALL EMITDATAWORD(0);  /*VALUE OF PARM */
            CALL FINDADDRESS(DP-4);
            SYBASE(NDECSY) = ADREG;
            SYDISP(NDECSY) = ADRDISP;
            SYESDTYPE(NDECSY) = "0F";  /* "0F" = EXTERNAL PARAMETER */
         END;
      ELSE
      IF ESDTYPE = "01" THEN    /* PUBLIC PROC PARM */
         DO;
            SYTCO(NDECSY) = PARCT;  /* LOCATION OF PARM IN LIST */
            SYESDTYPE(NDECSY) = "0E";  /* USED LATER IN DECLARE GRAMMAR*/
         END;

   END;

/*  <ENDING> ::= END    */
   VAR(MP) = X0;

/*  <ENDING> ::= END <IDENTIFIER>    */
   VAR(MP) = VAR(SP);

/*  <ENDING> ::= <LABEL DEFINITION> <ENDING>    */
   VAR(MP) = VAR(SP);

/*  <LABEL DEFINITION> ::= <IDENTIFIER> :    */
   IF LENGTH(VAR(MP)) > 8 THEN  /* NOT SPECIAL LABEL */
       FIXL(MP) = ENTER (VAR(MP), LABELTYPE, PP, FIXL(MP));
   ELSE
       DO;
          IF VAR(MP) = 'EXTERNAL'  THEN
             ESDTYPE = "02" ;   /* TURN ON EXTERNAL BIT */
          ELSE
          IF VAR(MP) = 'PUBLIC' THEN
             ESDTYPE = "01" ;         /* IDENTIFY AS A PUBLIC PROCEDURE */
          ELSE
          IF VAR(MP) = 'COMMON' THEN
             ESDTYPE = "05";          /* LABELED COMMON */
```

```
         ELSE
            FIXL(MP) = ENTER (VAR(MP), LABELTYPE, PP, FIXL(MP));
      END;


/*  <RETURN STATEMENT> ::= RETURN    */
  CALL EMIT_RETURN;

/*  <RETURN STATEMENT> ::= RETURN <EXPRESSION>    */
  DO;  /* EMIT A RETURN BRANCH & PASS VALUE IN REGISTER 3 */
     /* NOW FORCE IT INTO REGISTER 3 */
     TARGET_REGISTER = 3;
     IF RETURNED_TYPE = CHRTYPE THEN
        CALL FORCEDESCRIPT(MPP1);
     ELSE
        CALL FORCEACCUMULATOR(MPP1);
     TARGET_REGISTER = -1;
     IF REG(MPP1) ~ = 3 THEN CALL EMITRR("18",3,REG(MPP1));
     CALL EMIT_RETURN;
     CALL CLEARREGS;
  END;

/*  <CALL STATEMENT> ::= CALL <VARIABLE>    */
  DO;
     CALL FORCEACCUMULATOR(SP);
     CALL CLEARREGS;
  END;

/*  <GO TO STATEMENT> ::= <GO TO> <IDENTIFIER>    */
  DO;
     CALL ID_LOOKUP(SP);
     J = FIXL(SP);
     IF J < 0 THEN           /*  1ST OCURRANCE OF THE LABEL */
        DO;
           CALL EMITDATAWORD(0);      /* SPACE FOR FIXUP */
           J = ENTER (VAR(SP), FORWARDTYPE, DP-4, FIXL(SP));
           SYTCO(J) = 1;
        END;
     IF SYTYPE(J) = LABELTYPE THEN
        CALL BRANCH_BD("F",SYBASE(J),SYDISP(J));
     ELSE IF SYTYPE(J) = FORWARDTYPE THEN
        DO;
           CALL EMITRX(LOAD,BRCHREG,0,SYBASE(J),SYDISP(J));
           CALL EMITRX(BC,"F",BRCHREG,PBR,0);
        END;
     ELSE
        DO;
           CALL ERROR('TARGET OF GO TO IS NOT A LABEL',0);
           CALL EMITRX(BC,"F",0,SYBASE(J),SYDISP(J));
        END;
  END;

/*  <GO TO> ::= GO TO    */
     ;
/*  <GO TO> ::= GOTO    */
     ;
```

```
/*  <DECLARATION STATEMENT> ::= DECLARE <DECLARATION ELEMENT>    */
    ;

/*  <DECLARATION STATEMENT> ::= <DECLARATION STATEMENT> , <DECLARATION ELEMENT>
        */
    ;

/*  <DECLARATION ELEMENT> ::= <TYPE DECLARATION>    */
  DO;
     IF TYPE(MP) = CHRTYPE THEN
        DSP = NEWDSP ;
     ELSE
        DO;
           DP = NEWDP ;
           CALL CHECKBASES ;
        END;
  END;

/*  <DECLARATION ELEMENT> ::= <IDENTIFIER> LITERALLY <STRING>    */
     IF TOP_MACRO >= MACRO_LIMIT THEN
        CALL ERROR('MACRO TABLE OVERFLOW',1);
     ELSE
        DO;
           TOP_MACRO = TOP_MACRO + 1;
           I = LENGTH(VAR(MP));
           J = MACRO_INDEX(I);
           DO L = 1 TO TOP_MACRO - J;
              K = TOP_MACRO - L;
              MACRO_NAME(K+1) = MACRO_NAME(K);
              MACRO_TEXT(K+1) = MACRO_TEXT(K);
           END;
           MACRO_NAME(J) = VAR(MP);
           MACRO_TEXT(J) = VAR(SP);
           DO J = I TO 255;
              MACRO_INDEX(J) = MACRO_INDEX(J)+1;
           END;
        END;

/*  <TYPE DECLARATION> ::= <IDENTIFIER SPECIFICATION> <TYPE>    */
  CALL TDECLARE(0);

/*  <TYPE DECLARATION> ::= <BOUND HEAD> <NUMBER> ) <TYPE>    */
  CALL TDECLARE(FIXV(MPP1));

/*  <TYPE DECLARATION> ::= <TYPE DECLARATION> <INITIAL LIST>    */
    ;

/*  <TYPE> ::= FIXED    */
  TYPE(MP) = FIXEDTYPE ;

/*  <TYPE> ::= CHARACTER    */
  TYPE(MP) = CHRTYPE ;

/*  <TYPE> ::= LABEL    */
  TYPE(MP) = FORWARDTYPE ;
```

```
/*  <TYPE> ::= <BIT HEAD> <NUMBER> )     */
     IF FIXV(MPP1) <= 8 THEN TYPE(MP) = BYTETYPE;
     ELSE IF FIXV(MPP1) <= 16 THEN TYPE(MP) = HALFWORD;
     ELSE IF FIXV(MPP1) <= 32 THEN TYPE(MP) = FIXEDTYPE;
     ELSE TYPE(MP) = CHRTYPE;

/*  <BIT HEAD> ::= BIT (    */
     ;

/*  <BOUND HEAD> ::= <IDENTIFIER SPECIFICATION> (    */
     ;

/*  <IDENTIFIER SPECIFICATION> ::= <IDENTIFIER>     */
  DO;
     INX(MP) = 1;
     I = FIXL(MP);
     FIXL(MP) = CASEP;
     IF CASEP >= CASELIMIT THEN
        CALL ERROR(DCLRM,1);
     ELSE
        CASEP = CASEP + 1;
     CASESTACK(CASEP) = ENTER (VAR(MP), 0, 0, I);
  END;

/*  <IDENTIFIER SPECIFICATION> ::= <IDENTIFIER LIST> <IDENTIFIER> )     */
  DO;
     INX(MP) = INX(MP) + 1;
     IF CASEP >= CASELIMIT THEN
        CALL ERROR(DCLRM, 1);
     ELSE
        CASEP = CASEP + 1;
     CASESTACK(CASEP) = ENTER (VAR(MPP1), 0, 0, FIXL(MPP1));
  END;

/*  <IDENTIFIER LIST> ::= (    */
  DO;
     INX(MP) = 0;
     FIXL(MP) = CASEP;
  END;

/*  <IDENTIFIER LIST> ::= <IDENTIFIER LIST> <IDENTIFIER> ,     */
  DO;
     INX(MP) = INX(MP) + 1;
     IF CASEP >= CASELIMIT THEN
        CALL ERROR(DCLRM, 1);
     ELSE
        CASEP = CASEP + 1;
     CASESTACK(CASEP) = ENTER (VAR(MPP1), 0, 0, FIXL(MPP1));
  END;

/*  <INITIAL LIST> ::= <INITIAL HEAD> <CONSTANT> )     */
  CALL SETINIT ;

/*  <INITIAL HEAD> ::= INITIAL (    */
  IF INX(MP-1) = 1 THEN
     ITYPE = TYPE(MP-1);     /*  INFORMATION FROM  <TYPE DECLARATION>  */
```

```
   ELSE
      DO;
         CALL ERROR('INITIAL MAY NOT BE USED WITH IDENTIFIER LIST',0);
         ITYPE = 0;
      END;


/*  <INITIAL HEAD> ::= <INITIAL HEAD> <CONSTANT> ,     */
   CALL SETINIT;

/*  <ASSIGNMENT> ::= <VARIABLE> <REPLACE> <EXPRESSION>     */
   CALL GENSTORE(MP,SP);

/*  <ASSIGNMENT> ::= <LEFT PART> <ASSIGNMENT>     */
   CALL GENSTORE(MP,SP);

/*  <REPLACE> ::= =     */
   ;

/*  <LEFT PART> ::= <VARIABLE> ,     */
   ;

/*  <EXPRESSION> ::= <LOGICAL FACTOR>     */
      ;
/*  <EXPRESSION> ::= <EXPRESSION> \ <LOGICAL FACTOR>     */
   /* "16" = OR, "56" = O */
   CALL ARITHEMIT("16");

/*  <LOGICAL FACTOR> ::= <LOGICAL SECONDARY>     */
      ;

/*  <LOGICAL FACTOR> ::= <LOGICAL FACTOR> & <LOGICAL SECONDARY>     */
   /* "14" = NR, "54" = N */
   CALL ARITHEMIT("14");

/*  <LOGICAL SECONDARY> ::= <LOGICAL PRIMARY>     */
   IF TYPE(MP) = CONDITION THEN CALL CONDTOREG(MP, REG(MP));

/*  <LOGICAL SECONDARY> ::= ^ <LOGICAL PRIMARY>     */
   DO;
      CALL MOVESTACKS (SP, MP);
      IF TYPE(MP) = CONDITION THEN
         CALL CONDTOREG (MP, "E" - REG(MP));
      ELSE
         DO;
            CALL FORCEACCUMULATOR (MP);
              /* "57" = X */
            CALL EMITRX ("57", REG(MP), 0, DBR, COMPLOC);
         END;
   END;

/*  <LOGICAL PRIMARY> ::= <STRING EXPRESSION>     */
      ;

   /*              CONDITION CODES      MASK
                   0  OPERANDS EQUAL    BIT 8
```

```
                1  FIRST OPERAND LO BIT 9
                2  FIRST OPERAND HI BIT 10                                    *

/*  <LOGICAL PRIMARY> ::= <STRING EXPRESSION> <RELATION> <STRING EXPRESSION>
        */
  DO;
     CALL ARITHEMIT(CMPRR);
     BASES(REG(MP)) = AVAIL;
     REG(MP) = REG(MPP1);
     TYPE(MP) = CONDITION;
  END;

/*  <RELATION> ::= =    */
  REG(MP) = 6;

/*  <RELATION> ::= <    */
  REG(MP) = 10;

/*  <RELATION> ::= >    */
  REG(MP) = 12;

/*  <RELATION> ::= ~ =    */
  REG(MP) = 8;

/*  <RELATION> ::= ~ <    */
  REG(MP) = 4;

/*  <RELATION> ::= ~ >    */
  REG(MP) = 2;

/*  <RELATION> ::= < =    */
  REG(MP) = 2;

/*  <RELATION> ::= > =    */
  REG(MP) = 4;

/*  <STRING EXPRESSION> ::= <ARITHMETIC EXPRESSION>    */
     ;

/*  <STRING EXPRESSION> ::= <STRING EXPRESSION> \\ <ARITHMETIC EXPRESSION>
        */
  DO; /* CATENATE TWO STRINGS */
     MOVEFLAG = FALSE; /*TURN OFF POSSIBLE MOVE OF COMMON STRINGS */
     CALL FORCEDESCRIPT(MP);
     CALL EMITRX(STORE,REG(MP),0,A1,A2);
     BASES(REG(MP)) = AVAIL;
     CALL FORCEDESCRIPT(SP);
     CALL EMITRX(STORE,REG(SP),0,B1,B2);
     BASES(REG(SP)) = AVAIL;
     CALL CALLSUB(0,CATENTRY,3,MP);
     /* ASSUME CATENATE IS IN THE 1ST PAGE */
     MOVEFLAG = TRUE;  /* TURN ON TO ENABLE MOVE OF COMMON STRINGS */

     TYPE(MP) = DESCRIPT;
  END;
```

```
/*  <ARITHMETIC EXPRESSION> ::= <TERM>    */
    ;
/*  <ARITHMETIC EXPRESSION> ::= <ARITHMETIC EXPRESSION> + <TERM>    */
  /* "1A" = AR, "5A" = A  */
  CALL ARITHEMIT("1A");

/*  <ARITHMETIC EXPRESSION> ::= <ARITHMETIC EXPRESSION> - <TERM>    */
  /* "1B" = SR, "5B" = S */
  CALL ARITHEMIT("1B");

/*  <ARITHMETIC EXPRESSION> ::= + <TERM>    */
  CALL MOVESTACKS(MPP1,MP);

/*  <ARITHMETIC EXPRESSION> ::= - <TERM>    */
  DO;
     CALL MOVESTACKS(MPP1, MP);
     IF TYPE(MP) = CONSTANT THEN FIXV(MP) = - FIXV(MP);
     ELSE
        DO;
           CALL FORCEACCUMULATOR(MP);
           CALL EMITRR("13", REG(MP), REG(MP));  /* LCR = COMPLEMENT */
        END;
  END;

/*  <TERM> ::= <PRIMARY>    */
    ;

/*  <TERM> ::= <TERM> * <PRIMARY>    */
  /* "1C" = MR, "5C" = M */
  DO;
     CALL FORCEACCUMULATOR(MP);
     IF REG(MP) = 1 THEN
        DO;  /* MULTIPLY IS FUNNY ON A 360--SORRY */
           REG(MP) = 0;
           CALL ARITHEMIT("1C");
           REG(MP) = 1;
        END;
     ELSE
        DO;
           CALL FORCEACCUMULATOR(SP);
            IF REG(SP) = 1 THEN
              DO;
                 CALL EMITRR("1C",0,REG(MP));
                 BASES(REG(MP)) = AVAIL;
                 REG(MP) = 1;
              END;
           ELSE IF REG(MP) + REG(SP) = 5 THEN
              DO;  /* OPERANDS ARE IN 2 & 3 */
                 CALL EMITRR("1C",2,2);
                 BASES(2) = AVAIL;
                 REG(MP) = 3;
              END;
           ELSE CALL ERROR ('MULTIPLY FAILED ***', 1);
        END;
  END;
```

```
/*  <TERM> ::= <TERM> / <PRIMARY>    */
  CALL DIVIDE_CODE;
  /*  DIVIDE IS EVEN FUNNIER THAN MULTIPLY  */

/*  <TERM> ::= <TERM> MOD <PRIMARY>    */
  DO;
     CALL DIVIDE_CODE;
     CALL EMITRR("18",1,0);                        /*  LR    1,0        */
  END;

/*  <PRIMARY> ::= <CONSTANT>    */
     ;
/*  <PRIMARY> ::= <VARIABLE>    */
     IF FIXV(MP) = 3 THEN   /*  FINISH OFF THE FUNCTION  BYTE        */
        IF CNT(MP) = 1 THEN
           DO;
              IF TYPE(MP) = CHRTYPE THEN
                 DO;
                    TYPE(MP) = CONSTANT;
                    FIXV(MP) = BYTE(VAR(MP));
                 END;
              ELSE
                 DO;
                    I = FINDAC;
                    CALL EMITRR("1B",I,I);       /*  SR    I,I        */
                    CALL EMITRX("43",I,0,REG(MP),0);
                                                 /*  IC               */
                    BASES(REG(MP)) = AVAIL;
                    REG(MP) = I;
                    TYPE(MP) = ACCUMULATOR;
                 END;
           END;
        ELSE IF CNT(MP) = 2 THEN
           DO;
              I = INX(MP);
                    CALL EMITRX("43",I,I,REG(MP),0);
                    BASES(REG(MP)) = AVAIL;
                    REG(MP) = I;
              TYPE(MP) = ACCUMULATOR;
           END;

/*  <PRIMARY> ::= ( <EXPRESSION> )    */
  CALL MOVESTACKS(MPP1, MP);

/*  <VARIABLE> ::= <IDENTIFIER>    */
  DO;       /* FIND THE IDENTIFIER IN THE SYMBOL TABLE */
     CALL ID_LOOKUP(MP);
     IF FIXL(MP) = -1 THEN
        CALL UNDECLARED_ID(MP);
  END;

/*  <VARIABLE> ::= <SUBSCRIPT HEAD> <EXPRESSION> )    */
  DO;  /* EITHER A PROCEDURE CALL OR ARRAY OR BUILT IN FUNCTION */
     CNT(MP) = CNT(MP) + 1;
     I = FIXV(MP);
```

```
IF I < 6 THEN
DO CASE I;

   /* CASE  0  */

   DO;       /* SUBS \ CALL */
      CALL FORCEACCUMULATOR (MPP1);
      IF SYTYPE(FIXL(MP)) = LABELTYPE \
         SYTYPE(FIXL(MP)) = CHAR_PROC_TYPE  THEN
         CALL STUFF_PARAMETER;
      ELSE
         DO;       /* SUBSCRIPTED VARIABLE */
            IF CNT(MP) > 1 THEN
               CALL ERROR ('MULTIPLE SUBSCRIPTS NOT ALLOWED', 0);
            INX(MP) = REG(MPP1);
         END;
   END;

   /* CASE  1  */

   DO;  /* BUILT IN FUNCTION: LENGTH */
      CALL FORCEDESCRIPT (MPP1);
      CALL EMITRR ("12", REG(MPP1), REG(MPP1));  /* LTR TO CHECK FOR NUL*/
      CALL EMITRX ("88", REG(MPP1), 0, 0, 24);  /* SHIFT TO CHARACTER */
      I = PP;
      CALL BRANCH (8, 0);   /* DON'T INCREMENT LENGTH ON NULL STRING */
      CALL EMITRX (LA, REG(MPP1), 0, REG(MPP1), 1); /*ADD 1, TRUE LENGTH*/
      CALL FIXBFW (I, PP);  /* DESTINATION OF NULL STRING JUMP */
      REG(MP) = REG(MPP1);  /* RECORD CONTAINING ACCUMULATOR */
      TYPE(MP) = ACCUMULATOR;
   END;

   /* CASE  2  */

   /* BUILT-IN FUNCTION SUBSTR */
   DO;
      IF CNT(MP) = 2 THEN
         DO;
            IF TYPE(MPP1) = CONSTANT THEN
               DO;
                  CALL EMITCONSTANT (SHL(FIXV(MPP1), 24) - FIXV(MPP1));
                  CALL EMITRX ("5F", REG(MP), 0, ADREG, ADRDISP);
               END;
            ELSE
               DO;
                  CALL FORCEACCUMULATOR (MPP1);
                  CALL EMITRR ("1E", REG(MP), REG(MPP1)); /* ALR BASE */
                  CALL EMITRX ("89", REG(MPP1), 0, 0, 24);
                  CALL EMITRR ("1F", REG(MP), REG(MPP1));
                  BASES(REG(MPP1)) = AVAIL;
               END;
            I = PP;
            CALL BRANCH (1, 0);     /* WE MAY NOW HAVE NEGATIVE LENGTH */
            CALL EMITRR ("1B", REG(MP), REG(MP));  /* NULL DESCRIPTOR */
            CALL FIXBFW (I, PP);
         END;
```

```
   ELSE
      DO;          /* THREE ARGUMENTS */
         CALL EMITRX (LA, REG(MP), INX(MP), REG(MP), PPSAVE(MP));
         BASES(INX(MP)) = AVAIL;
         IF TYPE(MPP1) ^= CONSTANT THEN
            DO;
               CALL FORCEACCUMULATOR (MPP1);
               CALL EMITRX (LA, REG(MPP1), 0, REG(MPP1), "FF");
                     /* DECREMENT LENGTH BY 1  */
               CALL EMITRX ("89", REG(MPP1), 0, 0, 24);
               CALL EMITRR ("16", REG(MP), REG(MPP1));  /* \ INTO D */
               BASES(REG(MPP1)) = AVAIL;
            END;
         ELSE
            DO;
               CALL EMITCONSTANT (SHL(FIXV(MPP1)-1, 24));
               CALL EMITRX ("56", REG(MP), 0, ADREG, ADRDISP);
            END;
      END;
   TYPE(MP) = DESCRIPT;
END;

/* CASE  3  */

DO;       /* BUILT IN FUNCTION BYTE */
   IF CNT(MP) = 1 THEN
      DO;
         IF TYPE(MPP1) = CHRTYPE THEN
            DO;
               TYPE(MP) = CHRTYPE;
               VAR(MP) = VAR(MPP1);
            END;
         ELSE
            DO;
               CALL FORCEDESCRIPT(MPP1);
               IF REG(MPP1) = 0 THEN
                  DO;
                     REG(MP) = FINDAC;
                     CALL EMITRR("18",REG(MP),0);
                                       /*  LR    REG(MP),0  */
                  END;
               ELSE
                  REG(MP) = REG(MPP1);
               TYPE(MP) = DESCRIPT;
               INX(MP) = 0;
            END;
      END;
   ELSE IF CNT(MP) = 2 THEN
      DO;
         CALL FORCEACCUMULATOR(MPP1);
         INX(MP) = REG(MPP1);
      END;
   ELSE
      CALL ERROR('BYTE CALLED WITH MORE THAN TWO ARGUMENTS',0);
END;
```

```
         /* CASE  4  */

         CALL SHIFT_CODE("89");          /*  SLL  */

         /* CASE  5  */

         CALL SHIFT_CODE("88");          /*  SRL  */
      END;      /* OF CASE STATEMENT */

      ELSE IF I = 10 THEN
         CALL EMIT_INLINE;
      ELSE IF I = 19 THEN     /*  BUILTIN  FUNCTION  ADDR   */
         DO;
            REG(MP) = FINDAC;
            CALL FORCE_ADDRESS(MPP1,REG(MP));
            TYPE(MP) = ACCUMULATOR;
         END;
      ELSE
         DO;
            CALL FORCEACCUMULATOR (MPP1);
            IF CNT(MP) = 1 THEN REG(MP) = REG(MPP1);
            ELSE INX(MP) = REG(MPP1);
         END;

   END;      /* OF PRODUCTION */

/*  <SUBSCRIPT HEAD> ::= <IDENTIFIER> (    */
   DO;
      CALL ID_LOOKUP(MP);
      IF FIXL(MP) < 0 THEN
         CALL UNDECLARED_ID(MP);
   END;

/*  <SUBSCRIPT HEAD> ::= <SUBSCRIPT HEAD> <EXPRESSION> ,     */
   DO;       /* BUILT IN FUNCTION OR PROCEDURE CALL */
      CNT(MP) = CNT(MP) + 1;
      IF FIXV(MP) = 0 THEN
         DO;       /* ~ BUILT IN FUNCTION */
            CALL FORCEACCUMULATOR (MPP1);
            IF SYTYPE(FIXL(MP)) = LABELTYPE \
               SYTYPE(FIXL(MP)) = CHAR_PROC_TYPE  THEN
               CALL STUFF_PARAMETER;
         END;
      ELSE IF FIXV(MP) = 2 \ FIXV(MP) = 3 THEN
         DO;       /* SUBSTR OR BYTE */
            IF CNT(MP) = 1 THEN
               DO;
                  CALL FORCEDESCRIPT (MPP1);
                  IF REG(MPP1) = 0 THEN
                     DO;
                        REG(MP) = FINDAC;
                        CALL EMITRR ("18", REG(MP), 0);
                     END;
                  ELSE REG(MP) = REG(MPP1);
               END;
            ELSE IF CNT(MP) = 2 THEN
```

```
                DO;
                   IF TYPE(MPP1) = CONSTANT THEN PPSAVE(MP) = FIXV(MPP1);
                   ELSE
                      DO;
                         CALL FORCEACCUMULATOR (MPP1);
                         INX(MP) = REG(MPP1);
                         PPSAVE(MP) = 0;
                      END;
                END;
             ELSE CALL ERROR ('TOO MANY ARGUMENTS TO SUBSTR \ BYTE');
          END;
       ELSE IF FIXV(MP) = 4 \ FIXV(MP) = 5 THEN
          DO;  /*  SHR  \  SHL  */
             CALL FORCEACCUMULATOR(MPP1);
             REG(MP) = REG(MPP1);
          END;
       ELSE IF FIXV(MP) = 10 THEN
          CALL EMIT_INLINE;
       ELSE IF FIXV(MP) >= 8 THEN
          DO;       /* SOME SORT OF MONITOR CALL */
             CALL FORCEACCUMULATOR (MPP1);
             IF CNT(MP) = 1 THEN REG(MP) = REG(MPP1);
             ELSE CALL ERROR ('TOO MANY ARGUMENTS FOR ' \\ SYT(FIXL(MP)));
          END;
       ELSE;       /* RESERVED FOR OTHER BUILT IN FUNCTIONS */
    END;

 /*  <CONSTANT> ::= <STRING>     */
    TYPE(MP) = CHRTYPE;

 /*  <CONSTANT> ::= <NUMBER>     */
    TYPE(MP) = CONSTANT;

    END;  /* OF CASE SELECTION ON PRODUCTION NUMBER */
END SYNTHESIZE;



  /*                SYNTACTIC PARSING FUNCTIONS                          */


RIGHT_CONFLICT:
   PROCEDURE (LEFT) BIT(1);
      DECLARE LEFT FIXED;
      /*  THIS PROCEDURE IS TRUE IF TOKEN IS ~ A LEGAL RIGHT CONTEXT OF LEFT*/
      RETURN ("C0" & SHL(BYTE(C1(LEFT), SHR(TOKEN,2)), SHL(TOKEN,1)
         & "06")) = 0;
   END RIGHT_CONFLICT;


RECOVER:
   PROCEDURE;
      /* IF THIS IS THE SECOND SUCCESSIVE CALL TO RECOVER, DISCARD ONE SYMBOL */
      IF ~ FAILSOFT THEN CALL SCAN;
      FAILSOFT = FALSE;
```

```
      DO WHILE ^ STOPIT(TOKEN);
         CALL SCAN;  /* TO FIND SOMETHING SOLID IN THE TEXT  */
      END;
      DO WHILE RIGHT_CONFLICT (PARSE_STACK(SP));
         IF SP > 2 THEN SP = SP - 1;  /* AND IN THE STACK  */
         ELSE CALL SCAN;  /* BUT DON'T GO TOO FAR  */
      END;
      OUTPUT = 'RESUME:' \\ SUBSTR(POINTER, TEXT_LIMIT+LB-CP+MARGIN_CHOP+7);
   END RECOVER;

STACKING:
   PROCEDURE BIT(1);  /* STACKING DECISION FUNCTION */
      COUNT#STACK = COUNT#STACK + 1;
      DO FOREVER;     /* UNTIL RETURN  */
         DO CASE SHR(BYTE(C1(PARSE_STACK(SP)),SHR(TOKEN,2)),SHL(3-TOKEN,1)&6)&3;

            /*  CASE 0  */
            DO;     /* ILLEGAL SYMBOL PAIR  */
               CALL ERROR('ILLEGAL SYMBOL PAIR: ' \\ V(PARSE_STACK(SP)) \\ X1 \\
                  V(TOKEN), 1);
               CALL STACK_DUMP;
               CALL RECOVER;
            END;

            /*  CASE 1  */

            RETURN TRUE;       /*  STACK TOKEN  */

            /*  CASE 2  */

            RETURN FALSE;      /* DON'T STACK IT YET  */

            /*  CASE 3  */

            DO;       /* MUST CHECK TRIPLES  */
               J = SHL(PARSE_STACK(SP-1), 16) + SHL(PARSE_STACK(SP), 8) + TOKEN;
               I = -1;  K = NC1TRIPLES + 1;  /* BINARY SEARCH OF TRIPLES  */
               DO WHILE I + 1 < K;
                  L = SHR(I+K, 1);
                  IF C1TRIPLES(L) > J THEN K = L;
                  ELSE IF C1TRIPLES(L) < J THEN I = L;
                  ELSE RETURN TRUE;  /* IT IS A VALID TRIPLE  */
               END;
               RETURN FALSE;
            END;

         END;     /* OF DO CASE  */
      END;   /*  OF DO FOREVER */
   END STACKING;

PR_OK:
   PROCEDURE(PRD) BIT(1);
      /* DECISION PROCEDURE FOR CONTEXT CHECK OF EQUAL OR IMBEDDED RIGHT PARTS*/
      DECLARE (H, I, J, PRD) FIXED;
      DO CASE CONTEXT_CASE(PRD);
```

```
         /*  CASE 0 -- NO CHECK REQUIRED  */

         RETURN TRUE;

         /*  CASE 1 -- RIGHT CONTEXT CHECK  */

         RETURN ^ RIGHT_CONFLICT (HDTB(PRD));

         /*  CASE 2 -- LEFT CONTEXT CHECK  */

         DO;
            H = HDTB(PRD) - NT;
            I = PARSE_STACK(SP - PRLENGTH(PRD));
            DO J = LEFT_INDEX(H-1) TO LEFT_INDEX(H) - 1;
               IF LEFT_CONTEXT(J) = I THEN RETURN TRUE;
            END;
            RETURN FALSE;
         END;

         /*  CASE 3 -- CHECK TRIPLES  */

         DO;
            H = HDTB(PRD) - NT;
            I = SHL(PARSE_STACK(SP - PRLENGTH(PRD)), 8) + TOKEN;
            DO J = TRIPLE_INDEX(H-1) TO TRIPLE_INDEX(H) - 1;
               IF CONTEXT_TRIPLE(J) = I THEN RETURN TRUE;
            END;
            RETURN FALSE;
         END;

      END;  /* OF DO CASE  */
   END PR_OK;


  /*                      ANALYSIS ALGORITHM                          */


REDUCE:
   PROCEDURE;
      DECLARE (I, J, PRD) FIXED;
      /* PACK STACK TOP INTO ONE WORD */
      DO I = SP - 4 TO SP - 1;
         J = SHL(J, 8) + PARSE_STACK(I);
      END;

      DO PRD = PR_INDEX(PARSE_STACK(SP)-1) TO PR_INDEX(PARSE_STACK(SP)) - 1;
         IF (PRMASK(PRLENGTH(PRD)) & J) = PRTB(PRD) THEN
            IF PR_OK(PRD) THEN
            DO;  /* AN ALLOWED REDUCTION */
               MP = SP - PRLENGTH(PRD) + 1; MPP1 = MP + 1;
               CALL SYNTHESIZE(PRDTB(PRD));
               SP = MP;
               PARSE_STACK(SP) = HDTB(PRD);
               RETURN;
            END;
```

```
      END;

      /* LOOK UP HAS FAILED, ERROR CONDITION */
      CALL ERROR('NO PRODUCTION IS APPLICABLE',1);
      CALL STACK_DUMP;
      FAILSOFT = FALSE;
      CALL RECOVER;
   END REDUCE;

COMPILATION_LOOP:
   PROCEDURE;

      COMPILING = TRUE;
      DO WHILE COMPILING;      /* ONCE AROUND FOR EACH PRODUCTION (REDUCTION)  */
         DO WHILE STACKING;
            SP = SP + 1;
            IF SP = STACKSIZE THEN
               DO;
                  CALL ERROR ('STACK OVERFLOW *** COMPILATION ABORTED ***', 2);
                  RETURN;    /* THUS ABORTING COMPILATION */
               END;
            PARSE_STACK(SP) = TOKEN;
            VAR(SP) = BCD;
            FIXV(SP) = NUMBER_VALUE;
            FIXL(SP) = CARD_COUNT;
            PPSAVE(SP) = PP;
            CALL SCAN;
         END;

         CALL REDUCE;
      END;      /* OF DO WHILE COMPILING  */
   END COMPILATION_LOOP;


  /*  MAIN PROCEDURES INVOLVED WITH THE CREATION OF THE SYSLIN FILE*/

INSERT_BINRCD:
   PROCEDURE(W,N);  /* INSERT 1-4 BYTES INTO THE BINARY OBJECT RECORD*/
     DECLARE(W,N,I,SHIFT) FIXED;

  /* CALCULATE THE INITIAL SHIFT COUNT I.E. SHIFT = (N-1) * 8  */
     SHIFT = SHL(N-1,3);
     DO I = 0 TO N-1;
        BINARY(BINPTR+I) = SHR(W,SHIFT);
        SHIFT = SHIFT - 8;
     END;
     BINPTR = BINPTR + N;
END INSERT_BINRCD;

  INIT_BINRCD:
    PROCEDURE(TYPE);
   DECLARE (TYPE,I) FIXED;
   DECLARE BINHDRS (12) BIT(8) INITIAL   ("C5","E2","C4",  /* ESD */
                                          "E3","E7","E3",  /* TXT */
                                          "D9","D3","C4",  /* RLD */
                                          "C5","D5","C4"); /* END */
```

```
   TYPE = TYPE * 3 - 3;  /* GET INDEX INTO BINHDRS */
   DO I = 1 TO 3;
      BINARY(I) = BINHDRS(TYPE);
      TYPE = TYPE + 1;
   END;
   DO I = 4 TO 79;  /* SET REST OF BINRCD TO SPACES */
      BINARY(I) = H40;
   END;
   IF TYPE < 10 THEN
       BINARY(10) = 0; /* SET BYTE COUNT HIGH ORDER BYTE TO 0 */
   BINPTR = 16;
  END INIT_BINRCD;



PUNCH_SYSLIN:
  PROCEDURE;
 DECLARE (I,J) FIXED;

   IF BINPTR - 16 < 1 THEN
      RETURN;                /* NOTHING TO WRITE */
   BINARY(11) = BINPTR - 16;     /* INSERT BYTE COUNT */

   SYSPUNCH = BINRCD ;
   DO I = 16 TO BINPTR;
      BINARY(I) = H40;
   END;
   BINPTR = 16;

END PUNCH_SYSLIN;


   OUTPUT_ESD:
     PROCEDURE(NAME,TYPE,ADDR,OTHER);
       DECLARE NAME CHARACTER;
       DECLARE TYPE BIT(8);
       DECLARE (ADDR, OTHER, I) FIXED;

       IF NAME = 'XPLINIT' THEN RETURN; /* NOT REALLY AN ENTRY */
        IF BINPTR > 63 THEN    /* THREE ESD'S PER CARD */
           DO;                /* WRITE A ESD RECORD TO SYSLIN */
               BINARY(14) = SHR(ESD_FIRST,8);
              BINARY(15) = ESD_FIRST;    /* INSERT FIRST NON LD TYPE*/
              CALL PUNCH_SYSLIN;      /* OUT TO OUTPUT(2) OR SYSLIN  */
              ESD_FIRST = "4040";
           END;
       NAME = PAD(NAME,8); /* FILL OUT TO 8 CHARACTERS */
       DO I = 0 TO 7 ;                /* GET THE ESD NAME */
          BINARY(BINPTR) = BYTE(NAME,I);
          BINPTR = BINPTR + 1;
       END;
       BINARY(BINPTR) = TYPE;
       IF TYPE ^= "01" & ESD_FIRST = "4040" THEN
          ESD_FIRST = ESDID;             /* FIRST NON LD ESDID */
       BINARY(BINPTR+1) = SHR(ADDR,16);
```

```
      BINARY(BINPTR+2) = SHR(ADDR,08);
      BINARY(BINPTR+3) = ADDR;          /* INSERT ADDR OR OFFSET*/
       BINPTR = BINPTR +5;      /* CNT FROM ABOVE & A COUPLE OF SPACES */
      BINARY(BINPTR) = SHR(OTHER,16);
      BINARY(BINPTR+1) =  SHR(OTHER,8);
      BINARY(BINPTR+2) = OTHER;
      BINPTR = BINPTR + 3 ;            /* SET PTR TO NEXT ESD ENTRY Z*/
     END OUTPUT_ESD;


PUNCH_ESD:
 PROCEDURE;

  DECLARE ESDBAD CHARACTER INITIAL('INVALID ESD TYPE, COMPILER BUG');
  DECLARE (ESDADDR,ESDOTHER) FIXED;



/* START BY EMITTING ESD'S FOR THE SECTIONS */

  IF MAIN_PTR ^= 0 THEN
     CHAR_TEMP = SYT(MAIN_PTR) \\ X4;  /* FIRST PUBLIC ENTRY */
  ELSE
     CHAR_TEMP = 'XPLMAIN ';       /* ROOT PROC NAME */
  DO I = 0 TO 6;
     IF BYTE(CHAR_TEMP,I) = BYTE(X1) THEN
         BYTE(CHAR_TEMP,I) = BYTE('*');  /* REPACE BLANKS WITH *'S */
  END;

  WORKBYTE = BYTE(CODE_ESDID);
  BYTE(CHAR_TEMP,7) = WORKBYTE;  /* APPEND ESDID */
  CALL OUTPUT_ESD(CHAR_TEMP,0,0,PP);

  WORKBYTE = BYTE(CODE_ESDID + 1);
  BYTE(CHAR_TEMP,7) = WORKBYTE;
  CALL OUTPUT_ESD(CHAR_TEMP,0,0,DP);

  WORKBYTE = BYTE(CODE_ESDID + 2);
  BYTE(CHAR_TEMP,7) = WORKBYTE;     /* STRINGS ESD NAME */
  ESDID = CODE_ESDID + 2;      /* LAST SYSTEM ESDID */
   ESD_FIRST = CODE_ESDID; /* FIRST ESDID IN THE ESD RECORDS */
  CALL OUTPUT_ESD(CHAR_TEMP,0,0,DESC(3)+256);

/* NOW CREATE ESD ENTRIES FOR THE REMAINING ENTRIES */

  CASEP,PARCT = 0;  /* USE CASE STACK TO HOLD POINTERS TO THE SYMBOL TABLE */
              /* TO BE USED LATER BY THE RLD PROCESSING MODULE */

  DO I = 0 TO NDECSY;  /* SEARCH SYMBOL TABLE */
     IF SYESDTYPE(I) ^= 0 THEN
        DO;
           CASEP = CASEP + 1;
           CASESTACK(CASEP) = I;  /* SAVE THE SYMBOL TABLE POINTER */
           ESDTYPE = SYESDTYPE(I);
           IF ESDTYPE = "01" THEN
```

```
            DO;
               ESDADDR = SHL(SYBASE(I),12) + SYDISP(I);
               ESDOTHER = CODE_ESDID;
            END;
         IF ESDTYPE = '02' THEN
            DO;
               ESDADDR = 0;
               ESDOTHER = '40404040';
               ESDID = ESDID + 1;
            END;
         IF ESDTYPE = '05' THEN
            DO;
               ESDADDR = 0;
               ESDOTHER = SYLEN(I);
               ESDID = ESDID + 1;
               IF SYTYPE(I) = CHRTYPE THEN /* SPECIAL CASE */
                  DO;
                     PARCT = PARCT + 1;
                     FIXCADR(PARCT) = I; /*  SYMBOL TABLE POINTER*/
                     ESDOTHER = SHL(ESDOTHER,5) + 276;
                  END;
            END;
          CALL OUTPUT_ESD(SYT(I),ESDTYPE,ESDADDR,ESDOTHER);
          SYTCO(I) = ESDID;  /* NEED ID FOR BUILDING RLD'S */
       END;
    END;

/* ALL ESD HAVE BEEN EMITTED CHECK FOR LEFTOVERS IN BUFFER */

  IF BINPTR > 16 THEN
     DO;
        BINARY(14) = SHR(ESD_FIRST,8);
        BINARY(15) = ESD_FIRST;
        CALL PUNCH_SYSLIN;
     END;
 END PUNCH_ESD;

PUNCH_TXT:
 PROCEDURE;
 DECLARE (I,BLOCK,BASEADDR) FIXED;
 DECLARE (TXT_ORG,TXT_END,TXT_ADDR,TXT_LIM) FIXED;
 DECLARE FILE_ID FIXED;


 DO I = 1 TO 4;
    BLOCK = 0;
    BASEADDR = ADDR(CODE);
    TXT_LIM = DISKBYTES;

    DO CASE I;

       ;        /* CASE 0 - NOTHING              */

       DO;      /* CASE 1 - CREATE CODE CSECT  */
          TXT_ORG = 0; TXT_END = PP;
          ESDID = CODE_ESDID;
```

```
            FILE_ID = I;
         END;

         DO;      /* CASE 2 - CREATE DATA CSECT   */
            TXT_ORG = 0; TXT_END = DP;
            ESDID = ESDID + 1;  /* ONE MORE THAN CODE ALWAYS */
            FILE_ID = I;
         END;

         DO;      /* CASE 3 - BUILD DESCRIPTOR FOR STRINGS */
            TXT_ORG = 0; TXT_END = DSP;
            TXT_LIM = DSP + 1;
            BASEADDR = ADDR (DESC); /* GET INFO FROM DESCRIPTOR ARRAY */
            ESDID = ESDID + 1;  /* ONE MORE THAN DATA ALWAYS */
         END;
         DO;      /* CASE 4 - CREATE STRINGS CSECT */
            TXT_ORG = DSP;  /* SAVE ROOM FOR DESCRIPTORS */
            TXT_END = DSP + CHP; /* DESCRIPTORS & STRINGS */
            TXT_LIM = TXT_LIM + TXT_ORG; /* FIX FOR BLOCK CONTRL */
            FILE_ID = 3;
         END;

      END;   /*  DO CASE   */

/* INSERT ESDID & CLEAR COL'S 17-72 ( 16-71 RELATIVE TO 0) */

      BINPTR = 5;
      CALL INSERT_BINRCD(TXT_ORG,3);
      BINPTR = 14;
      CALL INSERT_BINRCD(ESDID,2);


/* READ IN THE FIRST BLOCK OF DATA OR STRINGS */
         IF I ^= 3 THEN
            CODE = FILE(FILE_ID,0);

/* CREATE THE ACTUAL TXT RECORDS HERE   */

         DO TXT_ADDR = TXT_ORG TO TXT_END;
            IF TXT_ADDR >= TXT_LIM  THEN
               DO;
                  BLOCK = BLOCK + 1;
                  CODE = FILE (FILE_ID,BLOCK);
                  BASEADDR = ADDR(CODE);
                  TXT_LIM = TXT_LIM + DISKBYTES;
               END;

            IF BINPTR > 71 THEN
               DO;
                  CALL PUNCH_SYSLIN;
                  TXT_ORG = TXT_ADDR;
                  BINPTR = 5;
                  CALL INSERT_BINRCD(TXT_ORG,3);
                  BINPTR = 16;
               END;
            BINARY(BINPTR) = COREBYTE(BASEADDR);
```

```
            BINPTR = BINPTR + 1;
            BASEADDR = BASEADDR + 1;
          END;   /* ALL TXT FOR THIS ESDID DONE */
          IF BINPTR > 16 THEN
              CALL PUNCH_SYSLIN;
      END;   /*  DO I = 1 TO 4  */
  END PUNCH_TXT;

PUNCH_RLD:
  PROCEDURE;
   /*  PROCEDURE TO BUILD THE RLD ENTRIES FOR SYSLIN */

   DECLARE (R_PTR,P_PTR) BIT(16); /* RELOCATION AND POSITION ESDID'S  */
   DECLARE FLAG BIT(8);  /* RELOCATION FLAG */
       DECLARE (LAST_R,LAST_P) BIT(16);
   DECLARE FLAG_PTR FIXED INITIAL(20);  /* POINTER TO FLAG IN BINRCD */
       DECLARE LAST_FLAG BIT (8); /* USED WHEN R_PTR = P_PTR ETC...*/
   DECLARE (ADDR, I, J) FIXED;

   OUTPUT_RLD:
     PROCEDURE;

       IF BINPTR > 64 THEN /* BUFFER MIGHT BE GETTING FULL */
          DO;
             BINARY(FLAG_PTR) = LAST_FLAG;
             CALL PUNCH_SYSLIN;
             LAST_R,LAST_P = 0; /* SET UP FOR ANOTHER RCD */
             FLAG_PTR = 20; /* POSITION OF FIRST FLAG */
             BINPTR = 15;
          END;
       IF LAST_R = R_PTR & LAST_P = P_PTR THEN
          BINARY(FLAG_PTR) = LAST_FLAG \ "01"; /* TURN ON BIT 0 */
       ELSE
          DO;  /* LONG RLD SEGMENT */
             BINPTR = BINPTR + 1;
             BINARY(FLAG_PTR) = LAST_FLAG;
             BINARY(BINPTR) = SHR(R_PTR,8);
             BINARY(BINPTR+1) = R_PTR;
             BINARY(BINPTR+2) = SHR(P_PTR,8);
             BINARY(BINPTR+3) = P_PTR;
             BINPTR = BINPTR + 3; /* GETS BUMPED AGAIN BELOW */
             LAST_R = R_PTR; LAST_P = P_PTR;
          END;
       BINPTR,FLAG_PTR = BINPTR + 1;
       LAST_FLAG = FLAG;  /* FLAG BYTE MIGHT CHANGE AND R & P NOT */
       BINARY(BINPTR+1) = SHR(ADDR,16); /* FLAG GETS SET LATER */
       BINARY(BINPTR+2) = SHR(ADDR,08);
       BINARY(BINPTR+3) = ADDR;
       BINPTR = BINPTR + 3; /* ONE LESS THAN ACTUAL SO WE CAN TRY TO GET
                               AT LEAST 4 MORE BYTES ON THE SYSLIN RCD
                               IF THE BINPTR = 63    */
    END OUTPUT_RLD;


     BINPTR = 15;  /* IN THIS PROC WE INCREMENT FIRST */
```

```
 /* FIRST CREATE THE RLD FOR THE ADDRESSING RELOCATION */

  R_PTR,P_PTR = CODE_ESDID + 1;  /* DATA CSECT */
  FLAG = "0C";  /* ADCONS, 4 BYTES LONG */
  DO I = 4 TO DBR;  /* RELOCATE DATA BASES */
     ADDR = BASEDATA + SHL(I,2);  /* BASEDATA + 16 = R4 ETC... */
     CALL OUTPUT_RLD;
  END;

 /* NOW SET UP THE ADCONS IN THE STRING AREA */

  R_PTR,P_PTR = CODE_ESDID + 2;  /* STRING ESDID */
  DO I = 1 TO 3;  /* SET UP STRING DESCRIPTOR PREFACE */
     ADDR = SHL(I,2);
     CALL OUTPUT_RLD;
  END;

/* RELOCATE CONSTANT STRINGS   */
  DO I = 5 TO SHR(DSP,2);     /* START AFTER THE PROLOGUE */
     IF DESC(I) = 0 THEN ;     /* SORRY NO NOT ON THIS TERMINAL */
     ELSE
        DO;
           ADDR = SHL(I,2);
           CALL OUTPUT_RLD;
        END;
   END;


/* SET UP STRING BASE INTO BASES AREA */
  FLAG = "1C";  /* V-TYPE ADDRESS, 4 BYTES IN LENGTH */
  P_PTR = CODE_ESDID + 1; /* INTO CODE CSECT */
  ADDR = BASEDATA + 52; /* OFFSET FOR THE STRING BASE REGISTER */
  CALL OUTPUT_RLD;

/* SET THE BASE ADDR OF THE PROGRAM INTO THE DATA AREA */
  R_PTR = CODE_ESDID;
  ADDR = BASEDATA + 48;
   CALL OUTPUT_RLD;

/* ALL BASE REGISTERS ARE RELOCATED, NOW SET UP THE "PUBLIC",
   "EXTERNAL", AND "COMMON" VARIABLES */

/* FIRST DO ALL THE ENTRY POINTS (I.E. "PUBLIC"  ); EACH DEFINITION
   OF A PUBLIC ENTRY WILL REQUIRE THE RELOCATING OF THE DATA CSECT
   BASE ADDRESS INTO THE CODE CSECT.  THIS IS REQUIRED FOR INITIALIZATION
   OF BASE REGISTERS UPON ENTRY INTO THE PROCEDURE */

  R_PTR = CODE_ESDID + 1;  /* DATA CSECT GOES INTO THE */
  P_PTR = CODE_ESDID ;     /* CODE CSECT AT THE ADDRESS SPECIFIED
                              AT SYRLDADDR IN THE SYMBOL TABLE */

  DO I = 1 TO CASEP;  /* THE ESD ROUTINE PLACED THE SYMBOL TABLE
                         POINTERS INTO CASESTACK TO REDUCE TIME SEARCHING*/
     IF SYESDTYPE(CASESTACK(I)) = "01" THEN /* WE GOT ONE */
        DO;
           ADDR = SYRLDADDR(CASESTACK(I));
```

```
            CALL OUTPUT_RLD;
         END;
      END;

   /* NOW WE CAN PROCESS THE "EXTERNAL" AND "COMMON" TYPES  */

      P_PTR = CODE_ESDID + 1;  /* DATA CSECT */
      DO I = 1 TO CASEP;
         J = CASESTACK(I);  /* WE WILL USE 'CASESTACK(I)' A LOT HERE */
         IF SYESDTYPE(J) ^= "01" THEN
            DO;
               ADDR = SYRLDADDR(J);
               R_PTR = SYTCO(J);     /* SET THE ESDID */
               CALL OUTPUT_RLD;
            END;
      END;

   /* WE ARE FINALLY DONE WITH THOSE DARN RLD'S */

   /* LET'S NOT FORGET THE LEFTOVERS IN THE SYSLIN BUFFER */


     IF BINPTR > 16 THEN
        DO;
           BINARY(FLAG_PTR) = LAST_FLAG;
           CALL PUNCH_SYSLIN;
        END;

   END PUNCH_RLD;

   PUNCH_END:
    PROCEDURE;
     BINPTR = 5; /* INSERT ENTRY ADDR */
     CALL INSERT_BINRCD(MAINLOC,3);

     BINARY(14) = SHR(CODE_ESDID,8);
     BINARY(15) = CODE_ESDID;
     OUTPUT(2) = BINRCD;  /* SYSPUNCH */

   END PUNCH_END;


   /* ROUTINE TO PUNCH CSECTS FOR THE COMMON STRINGS WHICH REQUIRE
      SPECIAL PROCESSING BECOUSE OF THE WAY STRINGS ARE HANDLED IN
      GENERAL UNDER THIS VERSION OF THE XPL COMPILER   */

  PUNCH_COMMON:
     PROCEDURE;
     DECLARE (I,J,LEN) FIXED;

     ESDID = 1;  /* ALL COMMON CSECTS ARE CREATED INDIVIDUALLY WITH
                                          AN ESDID OF 1          */
     DO I = 1 TO PARCT;
        J = FIXCADR(I); /* SYMBOL TABLE POINTER */

    /* ALLOCATION ALGORITHM IS AS FOLLOWS:
```

```
   TOTAL CSECT LENGTH WILL BE
   # OF DESCRIPTORS * 4          "SYLEN(J)"
   # OF DESCRIPTORS * 128        "SHL(SYLEN..."
   PLUS 256 BYTES FOR PAD ABOVE FREELIMIT
   PLUS 20 BYTES FOR THE SYSTEM STRING PROLOGUE
*/

    LEN = SYLEN(J) + SHL(SYLEN(J),5) + 256 + 20;
    CALL INIT_BINRCD(1);  /* ESD SETUP */
    BINPTR = 14;
    CALL INSERT_BINRCD(ESDID,2);
    CALL OUTPUT_ESD (SYT(J),0,0,LEN);
    CALL PUNCH_SYSLIN;

/* TEXT CONTAINS THE FOLLOWING :
   WORD 0 = 0 (WILL CONTAIN THE DESCRIPTOR OF THE LAST STING CREATED
   WORD 1 = PREDEFINED TOP OF THE STRING AREA (FREEPOINT)
   WORD 2 = FREEBASE
   WORD 3 = MAXIMUM TOP ADDR OF THIS COMMON (FREELIMIT)
   WORD 4 = THE ACTUAL NUMBER OF STRING DESCRIPTORS IN THIS COMMON

   THE ABOVE AREA REQUIRES SPACE OF THE FIRST 4 DESCRIPTORS
   THEREFORE THE FIRST USER DESCRIPTOR IS DESC(5) WHICH IS
   SET IN THE INITIALIZATION ROUTINES
*/

    CALL INIT_BINRCD(2); /* INIT FOR TXT */
    BINPTR = 5;
    CALL INSERT_BINRCD(0,3);  /*SET ORIGIN TO 000 */
    BINPTR = 14;
    CALL INSERT_BINRCD(ESDID,2); /* INSERT ESDID */
    CALL INSERT_BINRCD(0,4);  /* WORD 0 */
    SP = SYLEN(J) + 20; /* FIRST BYTE FOR NEW STRINGS */
/* SYLEN(I) = # OF DESCRIPTORS * 4     */
    CALL INSERT_BINRCD(SP,4);  /* WORD 1 */
    CALL INSERT_BINRCD(SP,4);  /* WORD 2 */
    CALL INSERT_BINRCD(LEN-256,4);  /* WORD 3 */
    CALL INSERT_BINRCD((SHR(SYLEN(J),2)+5),4);  /* WORD 4 */

/* ZERO OUT THE REMAINING DESCRIPTOR POSITIONS  */
/* THIS IS NOT DONE BY THE LINKAGE EDITOR SO WE DO IT*/

    DO SP = 1 TO SHR(SYLEN(J),2);
       CALL INSERT_BINRCD(0,4);
       IF BINPTR > 71 THEN
          DO;
             CALL PUNCH_SYSLIN;
             BINPTR = 5;
             CALL INSERT_BINRCD(SHL(SP,2)+20,3);
             BINPTR = 16;
          END;
    END;

    CALL PUNCH_SYSLIN;

/* RELOCATE TXT WORDS 1 - 3 WITH THE ADDR OF THE COMMON STRING */
```

```
      CALL INIT_BINRCD(3);  /* RLD */
      LEN = '0D';  /* RLD FLAG BYTE - ADCON, 4 BYTES, R & P ARE = */
      CALL INSERT_BINRCD(SHL(ESDID,16)+ESDID,4); /* R & P POINTERS */
      DO SP = 1 TO 3;
         CALL INSERT_BINRCD(SHL(LEN,24)+SHL(SP,2),4);
      END;
      CALL PUNCH_SYSLIN;

  /* NOW WRITE THE END RECORD */

      CALL INIT_BINRCD(4);  /*  END  */
      SYSPUNCH = BINRCD;
   END;
END PUNCH_COMMON;


 EMIT_SYSLIN:
   PROCEDURE;
   DECLARE I FIXED;

   IF SEVERE_ERRORS > 0 THEN
      DO;
         OUTPUT = '***** SEVERE ERRORS DETECTED - NO SYSLIN FILE CREATED';
         RETURN;
      END;
   CALL INSERT_CODE_FIXUPS;
   FILE(STRINGFILE,CURSBLK) = STRINGS;
   FILE(DATAFILE,CURDBLK) = DATA;


  /* FIRST SET UP A COMMON LINK BETWEEN THE CHARACTER STRING BINRCD
     AND THE BYTE ARRAY 'BINARY' SO WE CAN INSERT BINARY INFO INTO
     'BINARY' AND USE 'BINRCD' TO OUTPUT TO THE CHARACTER ORIENTED
     FILE 'OUTPUT(2)' WHICH IS THE SYSLIN OUTPUT FILE FOR THE
     LINKAGE EDITOR 'SYSPUNCH'

     THE METHOD IS TO CREATE A STRING DESCRIPTOR IN THE LOCATION
     FOR 'BINRCD' WHICH CONTAINS THE ADDRESS OF 'BINARY'
     AND A LENGTH OF 79 (THE LENGTH GETS INCREMENTED IN THE OUTPUT
     ROUTINES); THEN THE ONLY REFERENCE TO 'BINRCD' SHOUL BE A
     'SYSPUNCH = BINRCD;'                                          */

     I = ADDR(BINRCD);  /* DESCRIPTOR ADDRESS */
     J = ADDR(BINARY);  /* ADDRESS OF BINARY ARRAY - BYTE 0 */
     COREBYTE(I) = 79;  /* HIGH ORDER BYTE IS LENGTH        */
     COREBYTE(I+1) = SHR(J,16); /* ADDRESS TOP BYTE */
     COREBYTE(I+2) = SHR(J,08); /* ADDRESS MIDDLE BYTE */
     COREBYTE(I+3) = J;         /* ADDRESS LOW BYTE    */


  /* THE ADDRESSES ARE NOW LINKED AND MAKES LIFE MUCH EASIER AND
     LESS TIME CONSUMING DURING CREATION OF THE SYSLIN DATA SET */

   BINARY(0) = '02';  /* ALL LOAD RECORDS START WITH HEX 02 */
```

```
   DO I = 1 TO 4; /* ESD, TXT, RLD, END */
      CALL INIT_BINRCD(I);
      DO CASE I;

         ;  /* CASE 0 - DOES NOT EXIST */

         CALL PUNCH_ESD; /* CREATE ESD RECORDS */

         CALL PUNCH_TXT; /* CREATE TXT RECORDS */

         CALL PUNCH_RLD; /* CREATE RLD RECORDS */

         CALL PUNCH_END; /* FINISH UP WITH END */

      END; /* DO CASE */
   END;
   IF PARCT > 0 THEN CALL PUNCH_COMMON;
 END EMIT_SYSLIN;


PRINT_SUMMARY:
   PROCEDURE;
      DECLARE I FIXED;
      CALL PRINT_DATE_AND_TIME ('END OF COMPILATION ', DATE, TIME);
      OUTPUT = X0;
      OUTPUT = CARD_COUNT \\ ' CARDS CONTAINING ' \\ STATEMENT_COUNT
         \\ ' STATEMENTS WERE COMPILED.';
      IF ERROR_COUNT = 0 THEN OUTPUT = 'NO ERRORS WERE DETECTED.';
      ELSE IF ERROR_COUNT > 1 THEN
         OUTPUT = ERROR_COUNT \\ ' ERRORS (' \\ SEVERE_ERRORS
            \\ ' SEVERE) WERE DETECTED.';
      ELSE IF SEVERE_ERRORS = 1 THEN OUTPUT = 'ONE SEVERE ERROR WAS DETECTED.';
         ELSE OUTPUT = 'ONE ERROR WAS DETECTED.';
      IF PREVIOUS_ERROR > 0 THEN
         OUTPUT = 'THE LAST DETECTED ERROR WAS ON LINE ' \\ PREVIOUS_ERROR
            \\ PERIOD;
      OUTPUT = PP \\ ' BYTES OF PROGRAM, ' \\ DP \\ ' OF DATA, ' \\ DSP
         \\ ' OF DESCRIPTORS, ' \\ CHP \\ ' OF STRINGS.  TOTAL CORE REQUIREMENT'
         \\ X1 \\ PP + DP + DSP + CHP \\ ' BYTES.';
      IF CONTROL(BYTE('D')) THEN CALL DUMPIT;
      CLOCK(3) = TIME;
      CALL EMIT_SYSLIN; /*CREATE THE SYSLIN FILE FOR LKED */
      DOUBLE_SPACE;
      CLOCK(4) = TIME;
      DO I = 1 TO 4;    /* WATCH OUT FOR MIDNIGHT */
         IF CLOCK(I) < CLOCK(I-1) THEN CLOCK(I) = CLOCK(I) +  8640000;
      END;
      CALL PRINT_TIME ('TOTAL TIME IN COMPILER   ', CLOCK(4) - CLOCK(0));
      CALL PRINT_TIME ('SET UP TIME              ', CLOCK(1) - CLOCK(0));
      CALL PRINT_TIME ('ACTUAL COMPILATION TIME  ', CLOCK(2) - CLOCK(1));
      CALL PRINT_TIME ('POST-COMPILATION TIME    ', CLOCK(4) - CLOCK(2));
      CALL PRINT_TIME ('SYSLIN CREATION TIME     ', CLOCK(4) - CLOCK(3));
      IF CLOCK(2) > CLOCK(1) THEN   /* WATCH OUT FOR CLOCK BEING OFF */
      OUTPUT = 'COMPILATION RATE: ' \\ 6000*CARD_COUNT/(CLOCK(2)-CLOCK(1))
         \\ ' CARDS PER MINUTE.';
   END PRINT_SUMMARY;
```

```
MAIN_PROCEDURE:
   PROCEDURE;
      CLOCK(0) = TIME;  /* KEEP TRACK OF TIME IN EXECUTION */
      CALL INITIALIZATION;

       CLOCK(1)  = TIME;
      CALL COMPILATION_LOOP;

      CLOCK(2) = TIME;
   EJECT_PAGE;

      /* CLOCK(3) GETS SET IN PRINT_SUMMARY */
      CALL PRINT_SUMMARY;

   END MAIN_PROCEDURE;


CALL MAIN_PROCEDURE;
RETURN SEVERE_ERRORS;

EOF EOF EOF
```

Bibliography

Aho, Alfred V. and Ullman, Jeffrey D., Principles of Compiler Design, 1977, Addison-Wesley Publishing Co.

Graham Robert M., Principles of Systems Programming, 1975, John Wiley & Sons Inc.

Gries, David, Compiler Construction for Digital Computers, 1971, John-Wiley & Sons Inc.

IBM Corporation, IBM System/360 Operating System: Job Control Language Reference Manual, GC28-6704-3, 1973.

IBM Corporation, IBM System/360 Operating System: Job Control Language User's Guide, GC28-6703, 1970.

IBM Corporation, OS PL/1 Optimizing Compilers: Language Reference Manual, GC-0009-4, 1976.

IBM Corporation, OS PL/1 Optimizing Compilers: Programmer's Guide, SC-0006, 1976.

IBM Corporation, OS PL/1 Optimizing Compilers: Program Logic Manual, LY33-6008, 1973.

IBM Corporation, OS/VS Linkage Editor and Loader, GC26-3813-4, 1975.

IBM Corporation, OS/VS Linkage Editor Program Logic Manual, SY26-3815-0, 1972.

IBM Corporation, Principles of Operation, GA22-7000-5, 1976

Intel Corporation, PL/M-80 Programming Manual, 98-268B, 1977.

Mckeeman, William M., Horning, James J, and Wortman, David B., A Compiler Generator, 1970, Prentice-Hall Inc.

Payne, William H., Machine Assembly and Systems Programming for the IBM 360, 1969, Harper & Row.