PROGRAM LISTING ORGANIZATION AND USAGE
COMMON SYSTEMS SOFTWARE DESCRIPTION
3A PROCESSOR

CONTENTS PAGE

1. GENERAL

1.01 This section describes the format and types of information contained in program listings (PRs) for the 3A Processor. It also provides an aid for using the program listings. In addition to this document, the following sections will aid in understanding and using the program listings:

SECTION	TITLE
232-160-100	Introduction to No. 2B Electronic Switching System, Software General Description, Generic EF-2
232-164-110	Program Listing Standards, Support Software Description, No. 2B Electronic Switching System
232-164-115	Basic Program Instruction Set Description, No. 2B Electronic Switching System
254-340-082	System Utilities, Software Subsystem Description, 3A Processor Extended Operating System
254-340-100	Introduction to 3A Language, 3A Processor Common Systems
254-340-102	Basic and Extended 3A Instruction Set, 3A Processor Common Systems

1.02 Whenever this section is reissued, the reason for reissue will be listed in this paragraph.

1.03 A list of abbreviations and acronyms used in this section is provided in Table A.

2. PROGRAM LISTING DESCRIPTION

PROGRAM DESIGN

2.01 Modular program design is used for the 3A Processor programs. The structure of a program is modular when the different tasks, subtasks, or functions are divided logically and packaged into relatively small program units which perform well-defined functions. Large programs [assembly unit or program identification (PIDENT), depending on the system] are divided into several small sections or program units (PUs) for ease of design, assembly, administration, maintenance, and readability. The assembly unit is the largest physical portion of a modular program, and is compiled or assembled as one entity. The assembly unit may or may not contain functionally related program units. A program unit is a subportion of a modular program and is a collection of code within an assembly unit which performs a well-defined function and is given a name by which it may be called. The programming techniques used also make use of pseudo-operations, macros, and subroutines in designing programs.

2.02 A pseudo-operation is an operation which may be used in an assembly statement to control assembler activities but does not result in executable code. These are instructions to the assembly program which control the structuring of information found in the listing. Pseudo-operations are an aid in writing and checking the program, in making storage-address assignments, and in defining data. PUBLIC, EXTERN, EQU, and EQUR(A) are examples of pseudo-operations.

2.03 A macro is a sequence of operations called by an abbreviated label or name. The macro can generate sequences of code depending on the parameters supplied in the macro call. The coded lines generated by the macro are easily identifiable in a listing by the macro level number which is surrounded by dashes, eg, -001- (macro level 1) or -002- (macro level 2). Macros may be contained within the PIDENT or may be contained in an external PIDENT referenced by the psuedo-operation GETLIB.

2.04 A subroutine is a block of code which provides a facility for maintaining a single copy of a given routine which can be used in a repetitive manner by other programs located in memory. Control of processing is passed from the calling program/subroutine to the called subroutine until the function is completed. Subsequently, control is returned to a location specified by the calling program. Subroutines may be either common or private subroutines. A common subroutine is one which has been declared PUBLIC; it can be used by a program specifying its use by an EXTERN pseudo-operation. It thus may be accessed by many programs. A private subroutine generally has application only in the PIDENT in which it is listed and is not declared as a common subroutine by a PUBLIC pseudo-operation; thus, is not accessible by other programs.

2.05 In using common subroutines, interface requirements must be satisfied. Verification of subroutine interfaces during assembly can be accomplished with macros. Various macros are provided by different systems to implement calling subroutines and returning from subroutines. These macros may have slightly different forms in the various applications; however, in most cases, the forms will incorporate the words CALL and RETURN either alone or with additional suffixes or prefixes added, eg, RETURNOS, S_CALL, SRETURN, etc.

Conversion Between Binary, Octal, and Hexadecimal Numbers

2.06 All digital computers use the binary system (two-state elements) internally to perform operations. A binary number is expressed in zeros and ones, such as: 1101. A binary digit is a number that can have one of two unique values; 0 or 1 and no other value. Binary digits can be used to represent numbers of any magnitude just as a string of decimal digits can be used to represent numbers greater than nine (9). A decimal digit can have one of ten unique values; 0 through 9. Decimal numbers cannot be easily handled by a computer, therefore decimal numbers are ultimately converted to binary numbers for use in a computer. Decimal numbers are called base ten numbers since there are ten unique values (0 through 9), and binary numbers are called base two numbers since there are only two unique binary values. Because binary numbers tend to be very long, binary numbers are often converted to another base number system. Conversion is accomplished by grouping the binary number into sets or groups of three of four binary digits and then converting to the other system. The numbering systems most often used are the base eight (three binary digits are used to represent eight unique values, 0 through 7), or base sixteen (four binary digits are used to represent sixteen unique values, 0 through 9 plus the letters A through F following after the digit 9). The octal or hexadecimal (base eight or base sixteen, respectively) numbers are often used rather than the base two (binary) numbers.

2.07 The 3A Processor uses either a 16-bit or 24-bit binary word, depending on the application. Therefore, a shorthand method called the octal number system is used (the hexadecimal number system is also used) to express fields in the listings. When each binary word is divided into 3-bit groups starting at the right, each group can be represented by one of the eight octal numerals 0, 1, 2, 3, 4, 5, 6, and 7.

2.08 Conversion between binary and octal numbers (base two to base eight) can be accomplished easily. The binary number is divided into sets or groups of three binary digits (beginning with the rightmost binary digit). Each position in the set of three represents a power of the numeral 2 whenever that position is occupied by a binary 1, and if a binary 0 occupies a position, it represents a zero. Hence the position of a binary 1 in the set of three binary digits represents a power of two that corresponds to the position of the binary digit 1. The rightmost position represents the zero power, the middle position represents the first power, and the leftmost position represents the second power of the numeral 2. The octal number is the sum of the powers of 2 denoted by the binary ones that are set in each group of three binary numbers when the word bits are divided into groups of three, beginning with the rightmost bit. For example, binary 001 represents the zero power of 2 which is octal number 1, binary 100 represents the second power of 2 which is octal number 4, and binary 111 represents the second, first, and zero powers of 2 which is octal number 7 (the sum of those 3 powers of 2, eg, 4 plus 2 plus 1). Binary - octal conversion is as follows:

Binary:

000 001 010 011 100 101 110 111

Octal:

0 1 2 3 4 5 6 7

2.09 Examples of conversions are given below:

Binary to octal:

Binary:

1 110 010 101 001 011

Octal:

1 6 2 5 1 3

Octal to binary:

Octal:

```
0 2 6 4 1 5
```

Binary:

```
0 010 110 100 001 101
```

2.10 Conversion between binary and hexadecimal (base sixteen) numbers is somewhat similar to the binary to octal conversion except; (1) the binary number is divided into sets of four binary digits rather than into sets of three binary digits, and (2) there are sixteen unique digits rather than eight as in the octal system. The position of a binary 1 in the set of four binary digits represents the power of two as before, except the highest power of two that can be represented by a binary 1 is now the third power of two rather than the second power. Table B provides a summary of the number system (binary, octal, hexadecimal, and decimal equivalents). The following is an example of a binary to hexadecimal conversion.

Binary:

```
0000 1100 0001 0010
```

Hexadecimal:

```
0 C 1 2
```

2.11 A symbolic instruction, its octal representation (as it would appear in the listing), and the binary or hexadecimal equivalent are represented typically as follows:

Symbolic instruction:

```
LR R1 R2
```

Octal representation:

```
006022
```

Binary equivalent:

```
0   C   1   2
0000110000010010.
```

Hexadecimal equivalent:

```
0C12
```

3A PROCESSOR PROGRAM LISTINGS

2.12 A program listing (PR) is a hard copy (printout) of a program and is produced for each segment of code assembled as a unit by the assembler program (this unit may also be referenced as a PIDENT). Each listing is identified by a name consisting of a maximum of six characters and a PR number (not necessarily unique). The name is normally an acronym which is functionally relevant to the listing. The PR number provides identification for the listing within a particular software system (series of programs). The program listing normally contains a prologue and comments which explain the function and logic of a particular listing in addition to recording the actual coded instructions which perform the functions.

2.13 A complete listing contains the following types of information. There is no absolute order in which the various types of information must appear; however, there is a general structure which can be followed. The various types of information found in the program listings are:

- Prologues and comments
- Control statements
- PUBLIC and EXTERN definitions
- Macro definitions
- Symbol definitions
- Data definitions
- Program section
- Symbol reference tables.

A. Prologues and Comments

2.14 Prologues and comments are placed in a listing to clarify and explain the function of the program instructions. There is no standard prologue format; thus the prologue may appear in various formats and contain a variety of information. Prologues explain why the program instructions are necessary, what function is accomplished, why it is done, and when it is done in relation to the other program instructions or functions. The prologue may be in table form with the what, why, when, etc, listed explicitly in tabular format, or the prologue may be a straight-forward narrative. Prologues are normally found at the beginning of a listing, but may be found throughout a listing. Comments are found throughout a listing and are used to explain one or more lines of code. Examples of various forms of prologues are shown in Fig. 1. Prologues and comments may be divided into four general categories:

(a) An assembly unit prologue which, when appropriate, serves to define the purpose and function of the assembly unit (PIDENT).

(b) A program unit prologue explains the function of well-defined functional program units within the assembly unit in greater detail than does the assembly unit prologue. The program unit prologues may be in various formats. Example prologues are shown in Fig. 2.

(c) A comment is used to explain a line (or lines) of code. A comment is denoted by a sharp sign (#) which precedes the comment. Comments may apply to several lines (a block) of code. A block of code is a small section of instructions identifiable as performing a subfunction or operation and may be given a name and considered as an entity. An example of a comment on a block of code is shown in Fig. 3.

(d) Comments may also be used on individual coded lines to explain the action of the code, data symbols, etc. An example of line comments is shown in Fig. 3.

B. *Control Statements*

2.15 Control statements are used to direct the activity of the assembler, identify the input, and meet certain assembler requirements such as record keeping, etc. Certain mandatory pseudo-operations are required to satisfy both the assembler and the loader. Various control statements (Fig. 4) normally appear at the beginning of the listing and establish the identity and version of the PIDENT. This is accomplished by use of the NAME pseudo-operation and the VERSION macro. The various libraries and macros which are required for use by the PIDENT are identified by various forms of the GETxxx pseudo-operation, eg, GETLIB, GETMAC, etc. The program listing may be one for an ESS Programming Language (EPL) program or one for a program which is written in assembler code. Control statements differ in the two cases:

(a) For an EPL program, the first EPL statement will normally appear with the other control statements. This statement, BEGIN_EPL, is used to initialize the EPL compiler and establish the various EPL options. Associated with the BEGIN_EPL statement is an END_EPL statement used to inform the compiler that no more EPL statements are to follow.

(b) Programs written in assembly code need not have the BEGIN_EPL statement.

(c) The END_EPL statement informs the compiler that no more EPL statements are to follow, ie, it is the limit of code to be assembled.

A detailed explanation of the control statements used in both types of listings is provided in Section 254-340-100.

C. PUBLIC, EXTERN, and DIRECT Definitions

2.16 The declaration of PUBLIC and EXTERN definitions (Figs. 5 and 6) normally appears at the beginning of the listing but is not restricted to this location and may appear elsewhere in the listing. The PUBLIC pseudo-operation is used to declare those symbols (such as subroutine names, buffer addresses, and entry points) in a program that can be referenced and used by other programs. Examples of PUBLIC statements are shown in Fig. 5. Comments may be included to explain the meaning of the statements.

2.17 The EXTERN pseudo-operation is used in a program to declare a symbol which has been defined and declared PUBLIC in another program but must be referenced by this program. Examples of EXTERN statements are shown in Fig. 6. Comments may be included with a declaration to explain the meaning of the symbol. As with the PUBLIC statement, the location of the EXTERN statement is not restricted to the beginning of the listing.

2.18 The DIRECT pseudo-operation is used to declare a symbol which has been defined in another program but must be referenced by this program. The DIRECT pseudo-operation is defined in the program listed in the location symbol (Fig. 6) field of the DIRECT statement (similar to the

EXTERN statements). Comments may be included with this statement to explain the meaning of the symbol.

D. Macro Definitions

2.19 When macros are defined in a program, the macro definition section may follow the PUBLIC and EXTERN definitions; however, the location is not restricted to the beginning of the listing. The macros that appear in a program listing are those used and/or controlled by that program to serve some specialized purpose. However, when a macro is used by more than one program, its definition is usually placed in a public macro library accessible by all programs to avoid duplication in coding or changing the macro. Each macro definition has the following elements:

(a) Macro name (MACRO)

(b) Parameters

(c) Dummy variables

(d) Functions and formula

(e) Macro end (MEND).

Comments, preceded by a sharp (#) sign, may be included to explain the function and purpose of the macro. An example of a macro is shown in Fig. 7.

E. Symbol Definition

2.20 In most cases pseudo-operations are used to define symbols when the symbol is placed in the location field. There is a collection of pseudo-operations (EQU, EQUR, EQURA, and SET) which are used solely for defining symbols. These may be used to equate two symbols, define program parameters, and assign symbolic values to general registers. This set of pseudo-operations normally appears after the PUBLIC and EXTERN definitions in the listing.

2.21 The EQU pseudo-operation permanently defines the symbol in the location field. Symbol is the name of the symbol to be defined and value is any expression whose value is assigned to symbol. Value may be a decimal, binary, octal, or hexadecimal number, or another symbol. This operation is used primarily for constants or program parameters.

```
LOCATION      OP     OPERAND
symbol        EQU    value
```

where value is the expression in the operand field. The SET pseudo-operation is identical to EQU except that it may be used to redefine a symbol without causing a multiple definition.

2.22 The 16 general registers are identified by numbers 0 through 15. However, the registers can be named symbolically and should be when this clarifies the use of the registers. The EQUR pseudo-operation is used to permanently define the symbols used for registers. Symbol is the name of the symbol to be defined by a decimal arithmetic expression value in the operand field and may have a numeric value of 0 through 15.

```
LOCATION      OP      OPERAND
symbol        EQUR    value
```

where value is any decimal arthmetic expression whose value is 0 through 15 inclusive.

2.23 Address register pairs RA0 and RA1 are defined symbolically by the EQURA pseudo-operation. Symbol in the location field is the name of the symbol which is defined by value in the operand field. The arithmetic expression value must be either 12 or 14 which will define the register pair 12 (register pair 12 and 13) or the register pair 14 (register pair 14 and 15) as the address register pair represented by symbol.

```
LOCATION      OP       OPERAND
symbol        EQURA    value
```

where value is any decimal arithmetic expression whose value is 12 or 14. Figure 8 provides examples of symbol definitions for registers.

2.24 The TEXT pseudo-operation is used to define a symbol to be equivalent to a string of text.

```
LOCATION     OP      OPERAND
symbol       TEST    "Text string"
```

where symbol is the TEXT symbol being defined and "Text string" are the characters to be used as the value or contents of the text symbol. The first and last quotes on the "Text string" serve as delimiters for the text string. The string itself may contain quotes.

2.25 The SYN (synonym) pseudo-operation is used to define a symbol (symbol1) as a synonym of another symbol (symbol2). The relationship is a lasting or permanent one even if the first symbol is redefined (using the SET pseudo-operation).

```
LOCATION     OP      OPERAND
symbol1      SYN     symbol2
```

where symbol1 is the name of the symbol being defined, and symbol2 is the name of the symbol synonymous with symbol 1.

2.26 The ARRAY pseudo-operation is used to define an array of arithmetic values and to assign an initial value to the array elements. The individual elements of an array may be assigned values by the SET or the EQU pseudo-operations. An element of an array may be used anywhere a symbol may be used in an expression. The total size of an array should not exceed approximately 850 elements, and the limit of the array dimensions is 10.

```
LOCATION   OP      OPERAND
           ARRAY   name    ([a1:] b1, . . . ,
                           [an:] bn) [=init], . .
```

where name is the name of the array, a1 is any decimal expression for the lower bound of the ith dimension, b1 is any decimal expression for the upper bound of the ith dimension, and init is the value assigned to each element of the array. The elements are referenced thus: name (ae1,...,aen) where ae1 is an arithmetic expression.

F. Data Definition Statements

2.27 The data definition statements (Fig. 9) are listed next in the program listing, and the statements are normally structured using pseudo-operations and system macros. These statements provide symbolic definitions and layouts of the primary data elements (data blocks and words) used in the program listing. A detailed layout of a memory table (Fig. 10) will include the length (number of words) of the data block and the width (number of bits) of partitions within the block. A partition may be a word, a number of consecutive words, a bit, or a number of consecutive bits within the block that are assigned meaningful names. The named partitions are handled as separate entities by the program. As a further aid to understanding the layout, a picture of the data block may follow immediately after the data definitions. The most common statements used for data definitions are the system macros LAYOUT or LOUT, ITEM, LOSKIP, and LOEND.

- LAYOUT or LOUT is used to characterize and name a data block by giving a name to the block and defining the number of consecutive words assigned to the block. A block may consist of one or more words.

- ITEM is used to characterize and name the partitions (words or bits within a word) within the data block defined by LAYOUT. A partition within a data block may be a word (16 bits or 24 bits per word depending upon the word length capability of the particular processor), a group of bits, or single bits within a word which are assigned meaningful names. The naming of the words or bits proceeds from the top of the item list to the bottom, and the naming of the bits proceeds from the rightmost to the leftmost bit within a 16-bit word. Thus, ITEM defines the width of the partitions of the data block proceeding from the first word location of the block to the last word location in an orderly manner.

- LOSKIP provides a means of automatically skipping words or bits in the data block (defined by LAYOUT) without counting the words or bits. The number of words to skip in the current block is defined or the number of bits to skip in the current word is defined.
- LOEND is used to indicate the end of the layout of the data block defined by LAYOUT or LOUT.

2.28 There is also a data definition pseudo-operation which is commonly used for addresses, ie, ADDR.

```
LOCATION    OP      OPERAND
symbol      ADDR    value
```

The ADDR pseudo-operation can be used to generate a 20-bit address in a double word. The symbol can be defined with a value equal to the address word. Value can be expressed either as a relocatable address or as a nonrelocatable decimal expression. A special use of ADDR is as the operand in an ITEM macro statement, where it will automatically allocate the 20 bits of address. The subsequent ITEM statements should then allocate the remaining 12 bits of the first word of the address pair.

2.29 Comments are often included in the data definition section to explain the use of the various data definitions. A comment normally follows the definition and is preceded by a sharp sign (#).

G. Program Section

CSECT and DATASECT

2.30 The program section contains the program instructions and comments describing the functions performed by the instructions. The program section may contain subsections called CSECT and DATASECT (Fig. 11). A CSECT is a pseudo-operation used to specify the beginning of a sequence of instructions which can be relocated as a block of instructions. A CSECT consists of a block of instructions that perform some well-defined function, identifies the beginning of an identifiable control unit, and assigns a location to the identity. CSECT is a mandatory pseudo-operation that must appear at the beginning of each control section in a program listing. Associated with the last CSECT is another mandatory pseudo-operation called END, which must appear at the end of the relocatable block comprising the assembly unit. The assembly unit may contain a number of program units (PUs) that perform some well-defined subfunctions of the assembly unit function. A DATASECT is a data CSECT used to declare areas of memory for the user program data. The DATASECT is given a label and appears at the beginning of the assembly unit.

2.31 A program unit prologue (Fig. 1) is normally found at the beginning of each program unit and will contain an explanation of the program unit. The format of the prologue is not standard but will generally be in one of the formats described previously, either the narrative or the tabular format.

2.32 An entry point is a labeled location at which a CSECT or a block of code may be entered and to which control may be transferred. An exit point is a location at which a block of code or CSECT terminates and gives control to some other location. Information about entry and exit conditions helps in the understanding of interrelationships between program units. Thus, comments may precede or accompany entry or exit points.

2.33 Paragraphs 2.34 through 2.52 refer to the examples of program listing pages within program sections. The title of the program unit is given at the top left corner of each page with a subtitle below it, if applicable. The assembly unit or PIDENT name may be given at the top right corner of each page. Across the bottom of the page is information such as the PIDENT name, time and date of the listing, the PR number, issue number, and page number.

2.34 Each program unit contains program instructions which instruct the 3A Processor to perform specific tasks. The program instructions are symbolic instructions which will be translated into binary machine instructions. Refer to Section 254-340-102 for explanations of the instructions.

2.35 The format of program instruction or line of code contains the following fields:

- Address field
- Encoded instruction field
- Branch address field
- Editor line number field or macro or EPL expansion level field
- Page line number field
- Location symbol field
- Symbolic operation code field
- Variable or operand field
- Comment field.

The fields are discussed in order of their appearance (left to right) on the program listing and are illustrated in Fig. 12.

Address Field

2.36 The address field (Fig. 12) is reserved for the relative (CSECT) address (memory location) of the program instruction. The address field contains the memory address (expressed in octal or hexadecimal numbers) of that particular line of code. When this field is blank, the line has been used for some purpose which does not require a memory location, eg, macro calls, comments, etc. A double word instruction will cause the address field to be incremented by two memory locations. A double word instruction at location 00053732 will cause the next printed location to be 00053734.

Encoded Instruction Field

2.37 The encoded instruction field entries are the octal or hexadecimal equivalent of the binary machine code, translated from a symbolic instruction. Only executable code entries appear in this field. No entries are present for a macro call, comments, or EPL statements, but are present for the macro or EPL expansions when they are listed. When expansions are suppressed in an EPL program, this field is blank (see Fig. 13) for all instructions to follow until the next EPL options statement is encountered that turns the EXPAND option on. The expansion is suppressed or not suppressed by setting the EXPAND option in the BEGIN_EPL statement to OFF or to ON respectively. For example, the statement

```
BEGIN_EPL EXPAND=OFF
```

suppresses the printout of the encoded instruction, whereas the statement

```
BEGIN_EPL EXPAND=ON
```

does not suppress the printout of the encoded instruction. There are variations in the encoded instruction field. These statements apply to the whole assembly unit. The EPL options statement EPLOPTIONS EXPAND = ON and EPLOPTIONS EXPAND = OFF are similar to the above but apply within an assembly unit rather than to the whole assembly unit. The field is different for the assembly code instructions, the 3A Processor instructions, and the 2B ESS instructions, (see Figs. 12 and 14).

2.38 For the assembly code (Fig. 12) the first digit in the encoded instruction field represents the branch-allowed (BA) bit. (The BA bit is equivalent to the TA bit used with the 2B ESS instructions.) When the BA bit is 1, a branch to that instruction from another location is allowed. When the BA bit is 0, a branch to that instruction is not allowed.

2.39 The octal or hexadecimal representation of the instruction follows the BA bit, and may be a single or a double word instruction. For example, the octal representation of the two-word instruction at address 00053741 (Fig. 12) is

```
1 007 00 01 020006
```

The 1 represents the BA bit, 007 00 01 represents the first word, and 020006 represents the second word. The 007 represents the OP code and 00 01 represents the Operand for the first word. The second word of the instruction is represented by 020006. The field for the second word is blank when the instruction is a one-word instruction.

The hexadecimal representation of the above two-word instruction is:

1 0701 2006

2.40 The 2-digit number following the absolute address (for both the 3A Processor and the No. 2B ESS instructions, Fig. 14) is the CSECT (control section) number. CSECT is a pseudo-operation which identifies the beginning or extension of a control section. The 6-digit octal number following the CSECT number is the location counter (offset) within the CSECT. Each time a CSECT pseudo-operation is encountered with a new name, the counter (associated with that CSECT name) is cleared. Every time a CSECT pseudo-operation is encountered with a previously established name, the location counter is set to the next unused location associated with that CSECT name.

2.41 For a 3A Processor instruction the encoded instruction field (Fig. 14) consists of two 6-digit octal numbers or one or two 4-digit hexadecimal numbers. The first 6-digit octal number is the first word of the encoded 3A CC instruction. The first digit in the number represents the branch-allowed (BA) bit. When the bit is equal to one, a branch to that instruction from another location is allowed; whereas, a zero means a branch to that instruction is not allowed. When an instruction is a 2-word instruction, a blank space and then another six digits are printed that represent the second word. The second octal expression is the second word of the 2-word instruction. This field is blank when the instruction is a one-word instruction.

2.42 For a 2B ESS instruction, the encoded instruction field consists of a 3-digit octal number and a 6-digit octal number (Fig. 14). The 3-digit number is the octal representation of the high eight bits of the 24-bit No. 2B ESS instruction. The 6-digit number is the octal representation of the low 16 bits of the instruction. The first digit in this 6-bit number represents the transfer-allowed (TA) bit (similar to the BA bit used with the 3A Processor instructions).

2.43 Refer to Sections 232-164-115 and 254-340-102 for a description of the basic instruction sets for the No. 2B ESS and the 3A Processor.

Branch Address Field

2.44 The branch address field contains the address of the location to which control is transferred by the execution of that line of instruction, or it contains the program name where a symbol is defined. The branch address field entries may be either octal or hexadecimal numbers or name entries. If the entry is an octal or hexadecimal number, the number is the transfer address in this same program listing to which control is transferred by the instruction. If the branch address field entry is a program name, the execution of the current instruction may transfer control to the named program; however, it may be that the current instruction does not transfer control to the named program. If the instruction uses a symbol defined in the named program, control is not transferred to the named program but the name is given of the other program where the symbol is defined. The entries in the branch address field are helpful, therefore, in identifying locations (addresses) in the current program or other programs to which control is transferred or in locating program listings where a symbol used in the current program is defined.

Editor Line Number Field

2.45 The editor line number (Fig. 12) is inserted by an editor program which may be used in the assembly of a program. This number is in decimal digits in a sequential order starting with 1 and increasing until the last line of the listing. The editor line number is interspersed with the macro or EPL expansion level number when the listing contains macro or EPL expansions. There is no line number when macros or EPL are not expanded in the listing.

Macro or EPL Level Number Field

2.46 When a macro is called, the sequence of program instructions defined by the macro is printed whenever the macro is expanded. The printing of the system macro expansion is controlled by the MACPRNT macro options ON and OFF. If ON is specified all system macros will be expanded, and if OFF is specified the system macros are not expanded (printed). The macro instructions are identified by a macro level number (Fig. 12). The level number identifies which level of nested macro generated this instruction line. The macro level number is preceded by and followed by a dash

(-XXX-); therefore, instructions generated by a macro and printed in the listing are easily identifiable, and replace the editor line number in the editor line column. The macro level number does not affect the sequence of the editor line numbers, but is interspersed with them.

2.47 Expansions of EPL statements are identified in the same manner as a macro expansion and have the same form.

Page Line Number Field

2.48 The line number (Fig. 12) contains two decimal digits which represent the line number on the program listing page. The line numbers begin with line 01 at the top of the page and may go up to the full page capacity of 50 lines.

Location Symbol Field

2.49 A location symbol is a symbolic name for a location by which branch instructions may refer to the instruction or to the block of code represented by the symbolic name. For instance, a branch instruction will refer to the symbolic name and control will be transferred to the named location.

Symbolic Operation Code Field

2.50 The symbolic operation (OP) code field contains the mnemonic or symbolic representation of the operation to be performed. It may be a basic instruction, an EPL operation, a macro name, or a pseudo-operation. The various operations are explained in Section 254-340-102 and Section 254-340-100.

Variable or Operand Field

2.51 The next field contains the variables or operands utilized by the operation. These may be in various forms, eg, symbolic, numerical, etc. (See Section 254-340-100.)

Comment Field

2.52 The comment field is not used in the assembly or execution of the program instructions, but is used only to note operations, describe, or explain the instructions. A comment is always preceded by a sharp (#) sign at the beginning of the comment. A comment may be located on a line to the right of an instruction or it may occupy a whole line itself. The sharp sign is used as a flag to indicate that no more executable code or assembly instruction follows the sign; however, whatever is entered after the sign is to be printed on the program listing as a comment.

H. Symbols Reference Tables

2.53 The table (sometimes called the Attributes and References Table) serves as an index into the program listings for the symbols (or names) that appear in that particular program listing. The table (Fig. 15) provides an easy access to the program pages and page line numbers where the symbol appears in the current listing and is referenced to an external program if the symbol is defined in the external program. A page and line number reference is also given to provide a ready reference to the page and line number where the symbol is defined. The symbols or names appear alphanumerically under the column named NAME. The information contained in the table is formatted under column headings with meaningful names. The column headings are: VALUE, T, NAME, DEF/REF, and ATTRIBUTES AND REFERENCES in the order of appearance (left to right) across the table.

2.54 The value of a symbol is listed under the heading VALUE within the symbol reference table. The equivalence of the symbol depends on the type of symbol and may be a numerical value, a PR name in which the symbol is defined, etc.

2.55 The type of symbol is listed under the heading T within the symbol reference list. The type (T) information is used to distinguish symbols of different types but having the same name. The meanings of the letters shown under the T heading are given in Table C.

2.56 The symbols are listed according to an alphanumerical collating sequence, regardless of length, under the column heading NAME. Numbers are listed after letters (letter A is low while letter Z is high followed by 0 through 9). Special characters are listed first, followed by lower case letters, upper case letters, and then numbers.

2.57 The page number and page line number at which a symbol is defined is listed under the heading DEF/REF, and the page and line number of each occurrence of a symbol is listed under the heading ATTRIBUTES AND REFERENCES. The page and line numbers are listed in pairs, eg, 36-15 which denotes page 36 of the listing and line number 15 on that page. The page and line numbers are listed for each appearance (or use) of the symbol in the listing. If a symbol is defined in the listing, the page and line number listed under the heading DEF/REF is the location where the symbol is defined.

2.58 If a symbol is defined in another program listing, the name of the other listing will be given under the heading VALUE. Page and line numbers where the symbol appears in the current listing will be given under the ATTRIBUTES AND REFERENCES heading. The first appearance of the symbol (defined in another listing) in the current listing will be given under the DEF/REF heading, but the definition will be given in the other program listing. Therefore, the information under the NAME, DEF/REF, and ATTRIBUTES AND REFERENCES headings provides easy access to definitions of and to each use of each symbol used in a program listing.

2.59 The Symbols Reference Table also contains information pertaining to symbols or names assigned to tables in memory and the layout or assignment of words and bits in the memory tables. A pictorial layout of a memory table (which may be one or more memory words) is sometimes provided in the program listing. The layout of the table is defined by a system layout macro. However, the table layout is also identified in the Symbols Reference Table by table name and item names, and by table size (TBLSIZ), width (W), beginning bit position or shift factor (S), and the offset of the word (N) in the table containing the item. Figure 10 illustrates a typical pictorial layout of a memory table (CART_LEN_ADDR) and the identification of the table layout in the Symbols Reference Table. The table name is given under the NAME heading, and a reference (for example page 5, line 02) to the table layout definition is given under the DEF/REF heading. The number of words in the table is given by TBLSIZ being equated to the number of words. Items defined as part of the table are identified by the table name appearing under the VALUE heading, and by the item name appearing under the NAME heading. The layout of the item in the table is given by the W, S, and N (width, bit position, and offset number of the word in the table, respectively) numbers given under the ATTRIBUTES AND REFERENCES heading. For example, item CART_LEN of CART_LEN_ADDR is 12 bits wide (W=C in hexadecimal), the first bit is located in bit position 4 (S=4), and is located in word 00 (N=0). The CL associated with each item is the class of the item, which is a decimal number that characterizes the table of which the item is a part.

OTHER PROGRAM LISTING FORMATS

2.60 Formats of program listings for the 3A Processor diagnostics are different from the format described herein because most processor diagnostics are table-driven routines. These tables are made up by various diagnostic test programs and are generated by a higher level macro language. The program section of the listings containing the data tables are different in several ways from the program section of listings described in this section; therefore, refer to Section 254-340-082 for information on the processor diagnostic listings. However, there are task monitor and interpretive routines which execute the diagnostics from the tables. These routines are in the 3A Processor language, and their program listing formats are similar to the format described in this section.

3. GLOSSARY

Assembly Unit—A collection of code that is assembled or compiled as one entity. The assembly unit is the largest physical portion of a modular program hierarchy, may or may not contain functionally related units, and is identified as a PIDENT.

ATTRIBUTES—Special characters reserved for specific functions or values which may be used only for the defined functions or values.

BEGIN EPL—The first EPL statement found in an EPL program listing; used to initialize EPL compiler and specify various options (see END_EPL).

CSECT—A pseudo-operation that identifies the beginning or extension of an identifiable assembly unit or control section and assigns a location name to the identity. A CSECT is relocatable as an entity.

Coded Line—Any statement in SWAP or EPL containing the defined LOCATION, OPERATION, VARIABLE, and COMMENT fields.

Comment Field—The field that contains notes to explain the instruction, operation, etc. In the listing a note is always preceded by a sharp (#) sign. Everything to the right of the sharp sign is taken as a comment.

DATASECT—A system macro used to declare and open a data CSECT (which is given a label or name) for data generated by system macros. A DATASECT is relocatable.

END EPL—The last EPL statement; used to inform the compiler that no more EPL statements are to follow (see BEGIN_EPL).

Entry Point—A labeled (named) location in an assembly unit, a program unit, a CSECT, or a block of code to which control may be transferred by a calling program, ie, the block of code may be entered at the labeled location.

Encoded Instruction—The octal or hexadecimal number representing the binary machine instructions and equivalent to the symbolic instruction.

Exit—A location at which a unit terminates and passes control to another location.

Location Field—The field of a formatted symbolic instruction which contains a label or name by which branch instructions may refer to the named instruction.

Macro—An instruction in EPL that is equivalent to a specified sequence of machine instructions, and is assigned a name by which it is called. When the macro is called, the name is expanded into the precoded lines of the macro routine. An active macro generates executable code while a declarative macro generates a data structure for an active macro.

Mnemonic—A functionally meaningful name assigned to represent various operations, data, locations, etc, or a combination of letters which convey the essence of an instruction (A for add, COM for complement, etc,).

Module—A small block of instructions within a CSECT which performs an identifiable subfunction.

NAME—A mandatory pseudo-operation which assigns a unique group of symbols to a PIDENT.

Op Code—The portion of an instruction which designates the operation that is to be performed (normally represented numerically). The op code specifies the operation that is to be performed upon the operand.

Operand—A value, label, or parameter to be processed by the associated op code.

Operand Field—The field of the instruction that contains parameters related to the operation and may contain predefined functions, symbolic variables, etc.

Operation—A code which directs the action of the EOS on other software, usually in a specific field of an instruction, and may be an operation code, pseudo-operation, macro, or other EPL constructions.

PIDENT NAME—A program identifier, which is a group of from 1 to 6 characters used to identify an assembly unit. A PIDENT number and title are normally associated with the PIDENT NAME.

PR Number—An identification number assigned to a PIDENT or to a collection of PIDENTS.

Program Listing—A printout of code assembled as a unit by the assembler program. This unit or program listing (PR) may also be referred to as a PIDENT.

Program Unit—A collection of code within an assembly unit which performs a well-defined function.

Pseudo-Operation—An operation which may be used by the assembler to control assembler activities but does not result in executable machine code.

References—The identification in the Attributes and References Table of the specific page number and line number where a symbol, label, name, or special character is used within an assembly unit or program unit. The page number is listed first and the line number follows. A dash (-) separates the page and line numbers.

Prologue—A narrative or tabular description of the assembly unit or program unit which is located at the beginning of the unit. The prologue tells the purpose of the unit and other general information about the unit.

Subroutine—A sequence of instructions called from within another section of instructions to perform a specific function. Subroutines may be common to several programs.

Symbol—A series of predefined characters or mnemonics which represent binary values.

```
#      THE TASK OF PROGRAM OPDATA IS TO UPDATE THE BACKUP FILES CONTAINING THE TRANSLATIONS FOR
#   THE NO. 3 ESS OFFICE.

#      EACH NO. 3 ESS OFFICE IS EQUIPPED WITH TWO TAPE DATA CONTROLLERS (TDCs 0 AND 1). EACH TDC
#   IN THE WORKING OFFICE SHOULD HAVE A DATA CARTRIDGE INSERTED IN THE TRANSPORT AT ALL TIMES, EXCEPT
#   WHEN MAINTENANCE FUNCTIONS REQUIRE ITS REMOVAL.  THE TWO CARTRIDGES ARE IDENTICAL, AND EACH ONE
#   CONTAINS SEVERAL FILES USED FOR VARIOUS PURPOSES.  THE MOST CRITICAL FILES ARE THOSE WHICH
#   ARE USED TO RELOAD MAIN STORE IN CASE A BOOTSTRAP IS DEEMED NECESSARY IN ORDER TO RETURN THE
#   SYSTEM TO AN OPERATIONAL STATE.  FOUR OF THESE FILES (CHECKSUM, TRNSLN, BACKDT1, AND BACKDT2) ARE
#   UPDATED BY THIS PROGRAM.  THE PROGRAM CONSISTS OF TWO SEPARATE PARTS: (1) THE UPDATING OF THE
#   MOST RECENT COPY OF TRANSLATIONS CONTAINED IN FILE TRNSL, AND (2) THE UPDATING OF THE BACKDATED
#   TRANSLATIONS IN FILE BACKDT1 OR BACKDT2.

#      IN MOST CASES IN WHICH A MEMORY RELOAD IS REQUIRED, RELOADING THE MOST RECENT COPY OF
#   TRANSLATIONS SHOULD BE SUFFICIENT TO RESTORE THE SYSTEM.  OPDATA IS INVOKED AUTOMATICALLY EVERY 24
#   HOURS TO UPDATE FILE TRNSLN AND THE ASSOCIATED DATA IN FILE CHECKSUM.  IT CAN ALSO BE INVOKED
#   MANUALLY WITH A TTY INPUT MESSAGE.  IT IS POSSIBLE THAT THE UPDATING PROCESS COULD ABORT DUE TO
#   AN I/O ERROR OR A REQUEST FROM THE MULTISCAN FUNCTION CONTROLLER (MSFC) TO ABORT.  THIS COULD
#   LEAVE THE OFFICE WITH A PARTIALLY UPDATED TRNSLN FILE.  A SUBSEQUENT MEMORY RELOAD WOULD LOAD THE
#   INVALID DATA INTO MAIN STORE.  TWO PRECAUTIONS ARE TAKEN TO PREVENT THIS FROM OCCURRING: (1) THE
#   UPDATE PROCESS TAKES PLACE ON ONE TAPE AT A TIME.  THEN, IF AN ABORT SHOULD OCCUR AND A MEMORY
#   RELOAD BECOMES NECESSARY BEFORE THE UPDATE CAN BE REPEATED SUCCESSFULLY, AT LEAST ONE TAPE WILL
#   CONTAIN GOOD TRANSLATION DATA.  (2) BEFORE THE UPDATE OF TRNSLN BEGINS, BIT 15 IN THE FIRST DATA
#   WORD OF FILE CHECKSUM IS SET.  THEN, IF THE UPDATE FAILS TO COMPLETE SUCCESSFULLY AND A MEMORY
#   RELOAD BECOMES NECESSARY BEFORE THE PROBLEM IS CORRECTED, THE BIT WILL INDICATE TO THE SYSTEM
#   INITIALIZATION PROGRAM THAT THE OTHER TAPE SHOULD BE USED FOR THE MEMORY RELOAD.

#      IT IS POSSIBLE THAT AN ERROR IN THE UP-TO-DATE TRANSLATION DATA COULD CAUSE A MEMORY RELOAD TO
#   BECOME NECESSARY.  IN THIS CASE, RELOADING THE CURRENT TRANSLATION DATA WOULD NOT CORRECT THE
#   PROBLEM.  IT WOULD BE NECESSARY TO RELOAD MEMORY WITH AN OLDER VERSION OF TRANSLATIONS AND DO
#   RECENT CHANGES TO BRING IT UP TO THE CORRECT, UP-TO-DATE STATUS.  THESE OLDER VERSIONS OF
#   TRANSLATIONS ARE KEPT IN FILES BACKDT1 AND BACKDT2.  OPDATA IS INVOKED MANUALLY BY TTY MESSAGE
#   TO DO THE UPDATES ON THESE FILES.  NO CHECKSUMS ARE ASSOCIATED WITH THE DATA ON THESE FILES, AND
#   THE UPDATE IS DONE ON BOTH TAPES AT ONCE. AGAIN, IF THE UPDATE WERE INTERRUPTED, A RELOAD WITH
#   BACKDATED TRANSLATIONS COULD RESULT IN THE LOADING OF ERRONEOUS DATA INTO MAIN STORE.  TO AVOID
#   THIS PROBLEM, OPDATA ZEROES ONE OF TWO SPECIAL WORDS IN FILE BACKDT1 (CONTAINING THE DATE OF LAST
#   UPDATE OF BACKDT1 AND BACKDT2, RESPECTIVELY).  IF THE FILES SHOULD BE ACCESSED DURING A MEMORY
#   RELOAD, A ZEROED DATE WORD INDICATES TO THE SYSTEM INITIALIZATION PROGRAMS THAT THE CORRESPONDING
#   FILE SHOULD NOT BE USED FOR THE RELOAD.
```

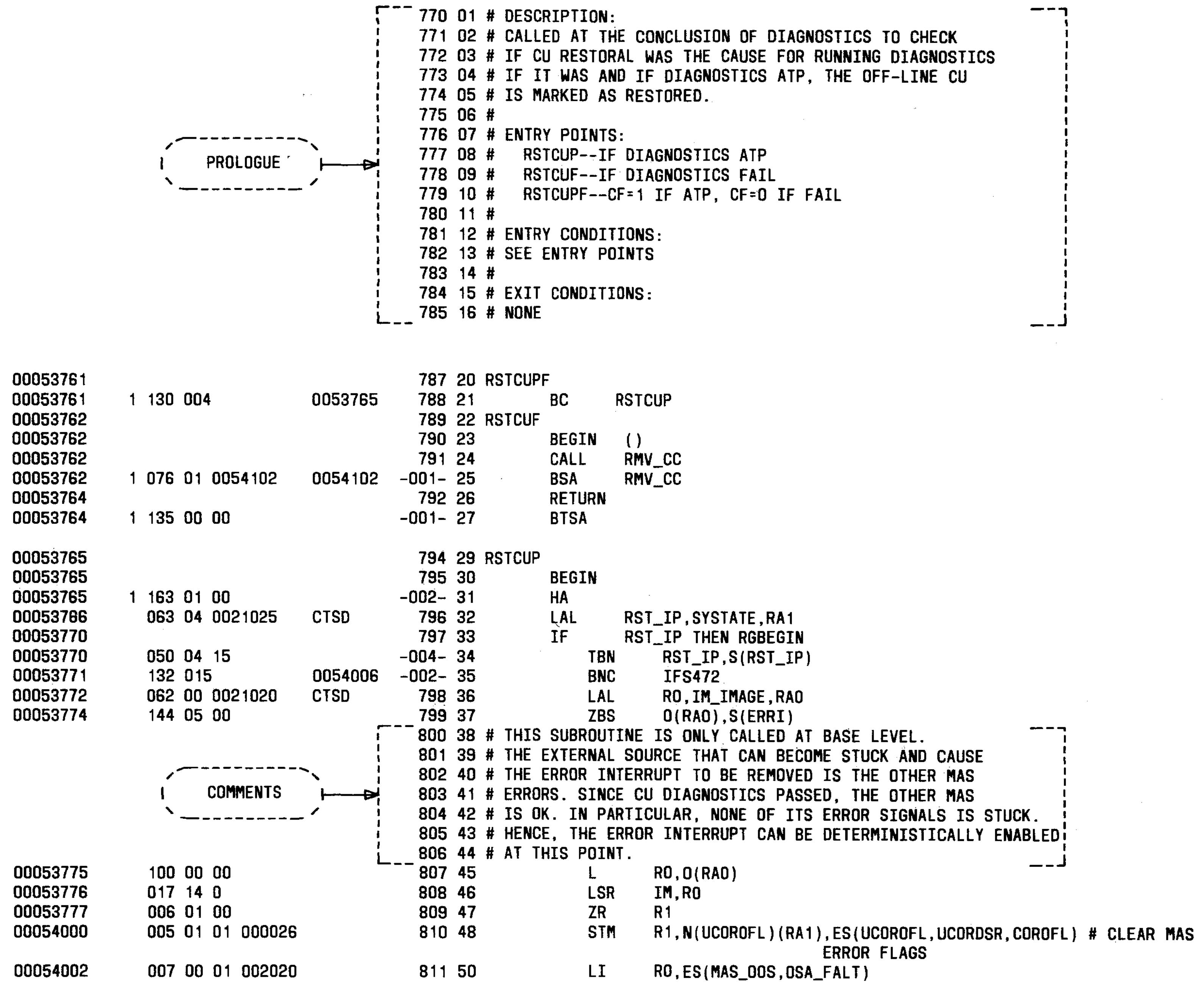
Fig. 1a—Assembly Unit Prologue Example

```
        NAME      MAICCI
#   THE FUNCTION OF THIS PROGRAM IS TO INITIALIZE THE EOS AND ASSOCIATED
#   TASKS.  IT MANY CASES, IT DOES NOT PERFORM THE ACTUAL INITIALIZATION,
#   BUT RATHER, CALLS THE APPROPRIATE INITIALIZATION CODE AS SUBROUTINES.


#   IF THE SYSTEM IS STARTED EITHER VIA A RAID ST OR CONT COMMAND, THE
#   SYSTEM IS PLACED IN THE DEBUG MODE.  THE DIFFERENCES BETWEEN DEBUG
#   AND NORMAL MODE ARE SLIGHT.  THEY ARE AS FOLLOWS:
#     (1)  THE INITIALIZATION DB COUNTING DISPLAY ONLY OCCURS IN THE DEBUG
#          MODE.
#     (2)  ON A START COMMAND, CINIT DOES NOT RETURN TO CIPL FOR TPE
#          FINAL PHASE OF THE BOOTSTRAP.  IT IS ASSUMED THAT THE SYSTEM HAS BEEN
#   LOADED FROM A 9TR.  CARE MUST BE TAKEN SO THAT THE SYSTEM NEVER GETS
#   INTO THIS MODE IN A WORKING OFFICE.

#   TWO WORDS ARE DEFINED IN MAIDTA WHICH INDICATE IF THE SYSTEM IS IN
#   THE DEBUG MODE.  THESE ARE SET TO ARBITRARY VALUES TO INDICATE
#   THE DEBUG MODE AS FOLLOWS:
#       MAIDEBUG ------ SET TO DEBUG IF SYSTEM WAS STARTED WITH A ST OR
#          CONT COMMAND.
#       MAISTART ------ SET TO START IF SYSTEM WAS STARTED WITH A ST COMMAND.
```

Fig. 1b—Assembly Unit Prologue Example

```
                              ASSEMBLY UNIT PROLOGUE

     NAME:     BUFFER


     TITLE:    BUFFER POOL MANIPULATION SUBROUTINES


     DOCUMENT: PR-73010 MISCELLANEOUS CALL PROCESSING
               BSP 230-100-216 COMMON BUFFER POOL


     THESE ROUTINES MANIPULATE THE COMMON BUFFER POOL AS FOLLOWS:
     ONE OR MORE BUFFERS MAY BE OBTAINED FROM THE AVAILABLE LIST
     (GET_BUF)
     ONE OR MORE BUFFERS MAY BE RELEASED TO THE AVAILABLE LIST
     (REL_BUF)
     AN ENTIRE MESSAGE MAY BE RELEASED TO THE AVAILABLE LIST
     (REL_MSG)
     A BUFFER MAY BE ADDED TO A MESSAGE AFTER AN INDICATED BUFFER
     (ADD_BUFA)
     A BUFFER MAY BE ADDED TO A MESSAGE BEFORE AN INDICATED
     BUFFER (ADD_BUFB)
     A BUFFER MAY BE REMOVED FROM A MESSAGE (RMV_BUF)
     A MESSAGE MAY BE ADDED TO A QUEUE AFTER AN INDICATED MESSAGE
     (ADD_MSGA)
     A MESSAGE MAY BE ADDED TO A QUEUE BEFORE AN INDICATED
     MESSAGE (ADD_MSGB)
     A MESSAGE MAY BE REMOVED FROM A QUEUE (RMV_MSG)
     NOTE:
     ALL NINE ROUTINES ARE NON-INTERRUPTIBLE AND COMPLETELY
     RE-ENTRANT
```

Fig. 1c—Assembly Unit Prologue Example



```
                                             770 01 # DESCRIPTION:
                                             771 02 # CALLED AT THE CONCLUSION OF DIAGNOSTICS TO CHECK
                                             772 03 # IF CU RESTORAL WAS THE CAUSE FOR RUNNING DIAGNOSTICS
                                             773 04 # IF IT WAS AND IF DIAGNOSTICS ATP, THE OFF-LINE CU
                                             774 05 # IS MARKED AS RESTORED.
                                             775 06 #
                                             776 07 # ENTRY POINTS:
                                             777 08 #   RSTCUP--IF DIAGNOSTICS ATP
                                             778 09 #   RSTCUF--IF DIAGNOSTICS FAIL
                                             779 10 #   RSTCUPF--CF=1 IF ATP, CF=0 IF FAIL
                                             780 11 #
                                             781 12 # ENTRY CONDITIONS:
                                             782 13 # SEE ENTRY POINTS
                                             783 14 #
                                             784 15 # EXIT CONDITIONS:
                                             785 16 # NONE

00053761                                     787 20 RSTCUPF
00053761   1 130 004           0053765       788 21        BC      RSTCUP
00053762                                     789 22 RSTCUF
00053762                                     790 23        BEGIN   ()
00053762                                     791 24        CALL    RMV_CC
00053762   1 076 01 0054102    0054102      -001- 25        BSA     RMV_CC
00053764                                     792 26        RETURN
00053764   1 135 00 00                      -001- 27        BTSA

00053765                                     794 29 RSTCUP
00053765                                     795 30        BEGIN
00053765   1 163 01 00                      -002- 31        HA
00053766     063 04 0021025    CTSD          796 32        LAL     RST_IP,SYSTATE,RA1
00053770                                     797 33        IF      RST_IP THEN RGBEGIN
00053770     050 04 15                      -004- 34            TBN     RST_IP,S(RST_IP)
00053771     132 015           0054006      -002- 35            BNC     IFS472
00053772     062 00 0021020    CTSD          798 36            LAL     RO,IM_IMAGE,RAO
00053774     144 05 00                       799 37            ZBS     O(RAO),S(ERRI)
                                             800 38 # THIS SUBROUTINE IS ONLY CALLED AT BASE LEVEL.
                                             801 39 # THE EXTERNAL SOURCE THAT CAN BECOME STUCK AND CAUSE
                                             802 40 # THE ERROR INTERRUPT TO BE REMOVED IS THE OTHER MAS
                                             803 41 # ERRORS. SINCE CU DIAGNOSTICS PASSED, THE OTHER MAS
                                             804 42 # IS OK. IN PARTICULAR, NONE OF ITS ERROR SIGNALS IS STUCK.
                                             805 43 # HENCE, THE ERROR INTERRUPT CAN BE DETERMINISTICALLY ENABLED
                                             806 44 # AT THIS POINT.
00053775     100 00 00                       807 45            L       RO,O(RAO)
00053776     017 14 0                        808 46            LSR     IM,RO
00053777     006 01 00                       809 47            ZR      R1
00054000     005 01 01 000026                810 48            STM     R1,N(UCOROFL)(RA1),ES(UCOROFL,UCORDSR,COROFL) # CLEAR MAS
                                                                                       ERROR FLAGS
00054002     007 00 01 002020                811 50            LI      RO,ES(MAS_OOS,OSA_FALT)
```

Fig. 2a—Program Unit Prologue Example

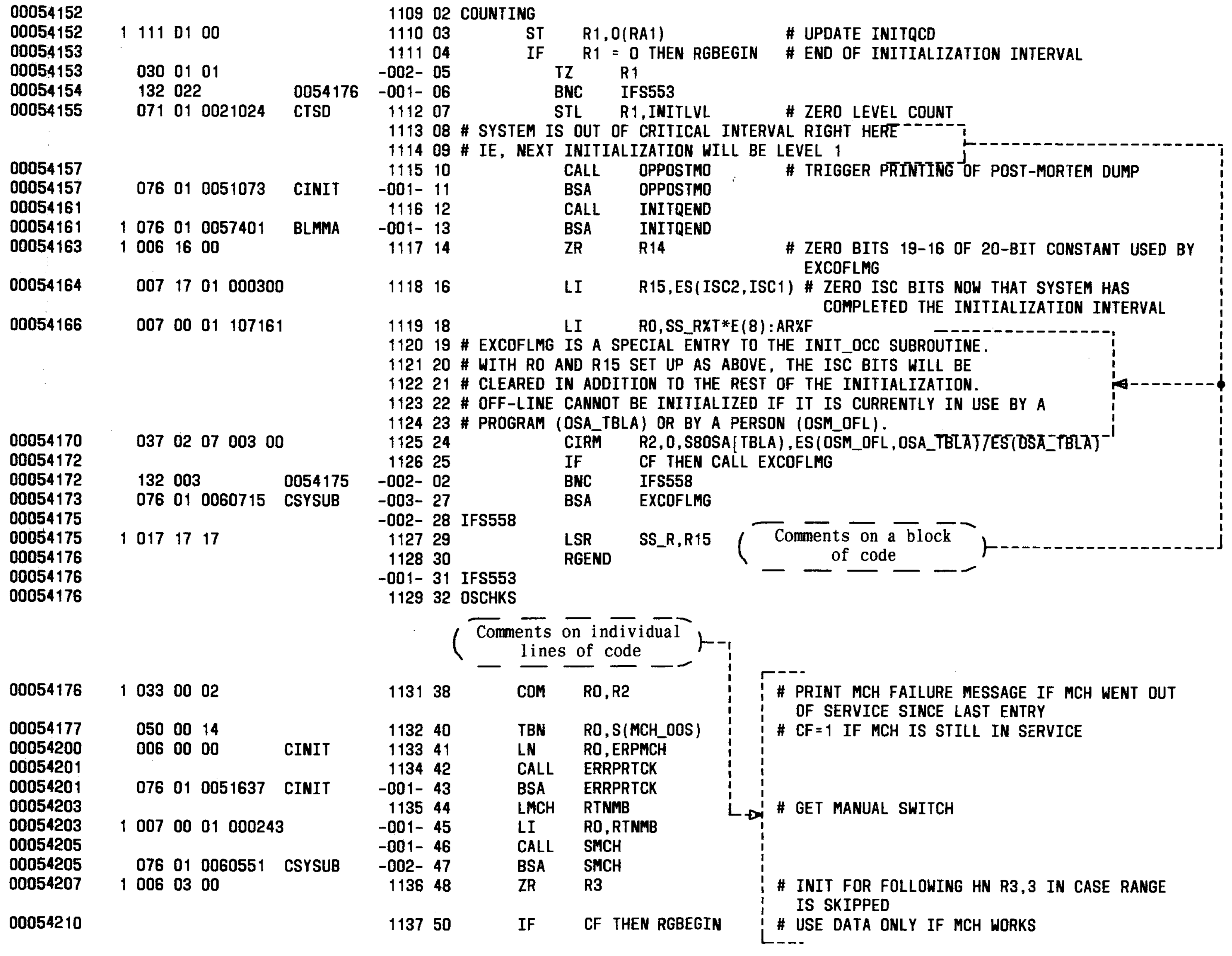
PROLOGUE

```
                                                    01 #*********************************************************************
                                                    02 # SUBROUTINE:SETFLG
                                                    03 #
                                                    04 #DESCRIPTION: THIS SUBROUTINE SETS THE SPECIFIED FLAG FOR THE SPECIFIED
                                                    05 #TASK. IT MAY BE CALLED ONLY FROM THE BACKGROUND LOOP WHICH COMPRISES
                                                    06 #MAINT.
                                                    07 #
                                                    08 #ENTRY CONDITIONS:
                                                    09 #     ENTRY VIA A BSA
                                                    10 #     R0=SYSTEM NAME OF TASK FOR WHICH EVENT FLAG IS TO BE SET
                                                    11 #     R1=EVENT FLAG NUMBER
                                                    12 #
                                                    13 #EXIT CONDITIONS:
                                                    14 #     REGISTERS ARE PRESERVED
                                                    15 #     CF=1 -> MESSAGE SENT OK
                                                    16 #     CF=0 -> NON EXISTENT PROCESS
                                                    17         PUBLIC  SETFLG
02 00304                                            18 SETFLG
02 00304    1 73 1 0                                19         HA
02 00305      CC 4 0                                20         LR      R4,R0
02 00306      CC 5 1                                21         LP      R5,R1
02 00307                                            22         SET_FLAG NAME=R4,SET=R5
02 00307      CC 1 4                          -002- 23         LR      R1,R4
02 00308      0C E C                          -001- 24         LR      R14,R12
02 00309      0C F D                          -001- 25         LR      R15,R13
02 0030A      0C D 5                          -002- 26         LR      R13,R5
02 0030B      07 C 1  0110                    -001- 27         LI      R12,X(110)
02 0030D      61 2 5                          -001- 28         SVC     5
02 0030E      56 04             02 00312      -001- 29         B       *+4
02 0030F      0C 00                           -001- 30         NOP
02 00310      0C 00                           -001- 31         NOP
02 00311      0C 00                           -001- 32         NOP
02 00312    1 0C C E                          -001- 33         LR     R12,R14
02 00313      0C D F                          -001- 34         LR     R13,R15
02 00314      5D 1 0                                35         ETSAG
```

Fig. 2b—Program Unit Prologue Example

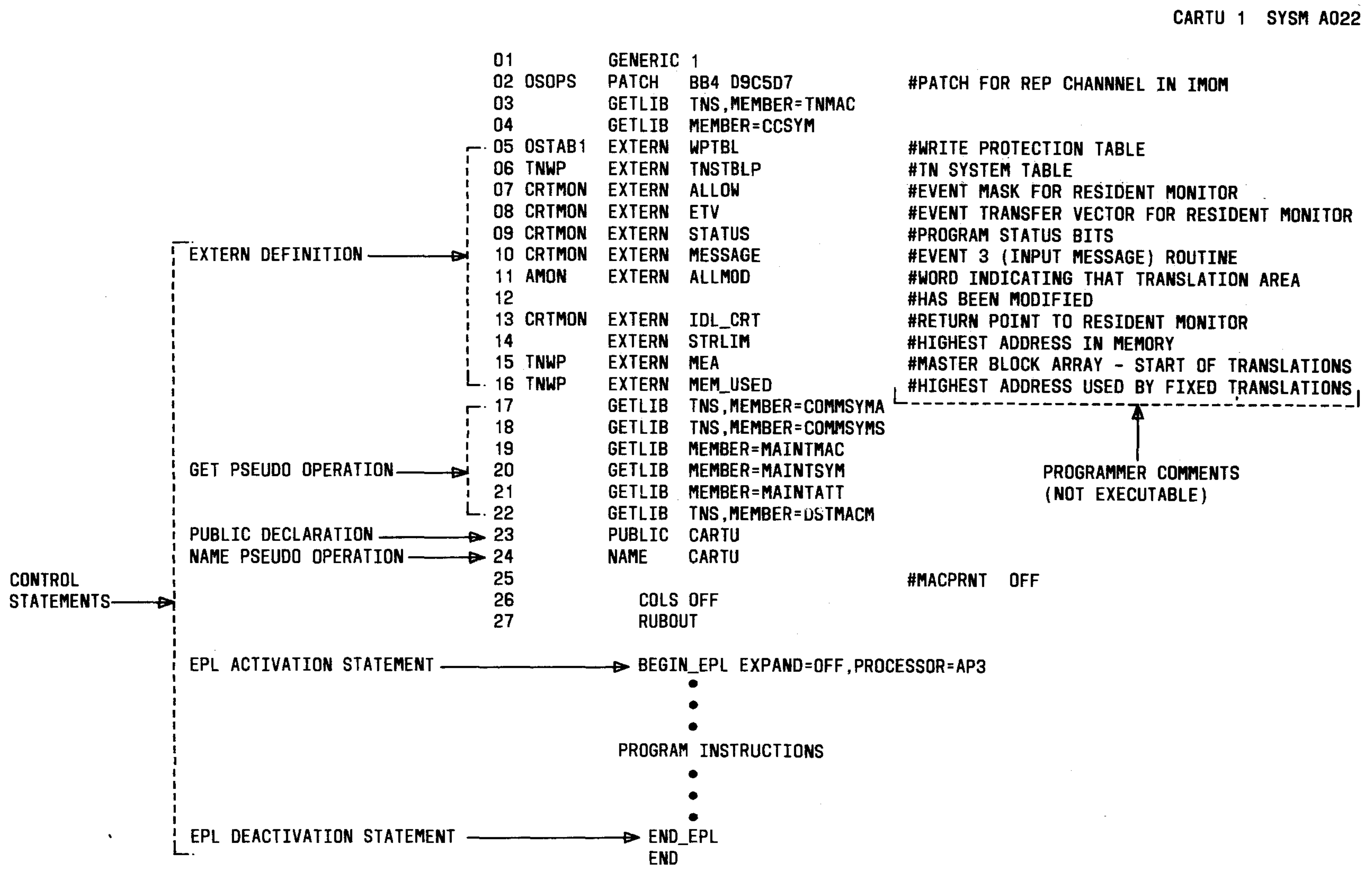
```
#***************************************************************************#
#                                                                           #
#                                 SYNC_CLI                                  #
#                                                                           #
#                       SYNCHRONOUS--CLEAR INTERRUPT                        #
#                                                                           #
#     WHAT:                                                                 #
#        THIS SUBROUTINE HANDLES UNANTICIPATED *IID* RESPONSES FROM BOTH    #
#        DEFINED AND UNDEFINED LINES                                        #
#                                                                           #
#     HOW:                                                                  #
#        DEFINED *SLA* ARE INITIALIZED AND THE *IIDTA* IS SET TO IGNORE     #
#        FUTURE ACTIVE *IID* RESPONSES                                      #
#                                                                           #
#     ENTRY:                                                                #
#       S_CALL SYNC_CLI                                                     #
#                                                                           #
#     ENTRY PARAMETERS:                                                     #
#       RP2 = SUBROUTINE RETURN ADDRESS                                     #
#       RP4 = ADDRESS OF *SLCBP* IF LINE IS DEFINED                         #
#       RP6 = ADDRESS OF *SLCBR* IF LINE IS DEFINED                         #
#       R8 = OFFSET INTO *SIIDTA* OF SUB-CHANNEL ADDRESSED BY *IID*         #
#       R9 = DEVICE ADDRESS OF *SLA* IF LINE IS DEFINED                     #
#       R12 = BIT OF ACTIVE *IID* RESPONSE                                  #
#                                                                           #
#     EXIT:                                                                 #
#      S_RETURN TO INSTRUCTION AFTER S_CALL                                 #
#                                                                           #
#     EXIT PARAMETERS:                                                      #
#       NONE                                                                #
#                                                                           #
#     INPUT/OUTPUT:                                                         #
#       TO *SLA*                                                            #
#                                                                           #
#     REGISTER USAGE:                                                       #
#       R0-R1 - UNUSED                                                      #
#       RP2 - SUBROUTINE RETURN ADDRESS AND EPL FREE                        #
#       R4-R7 - SAME AS ENTRY                                               #
#       R8 - SAME AS ENTRY                                                  #
#       R9 - SAME AS ENTRY                                                  #
#       R10-R11 - UNUSED                                                    #
#       R12 - SAME AS ENTRY                                                 #
#       R13 - UNUSED                                                        #
#       RA1 - EPL FREE                                                      #
#                                                                           #
#***************************************************************************#
```

Fig. 2c—Program Unit Prologue Example


Comments on a block of code
Comments on individual lines of code

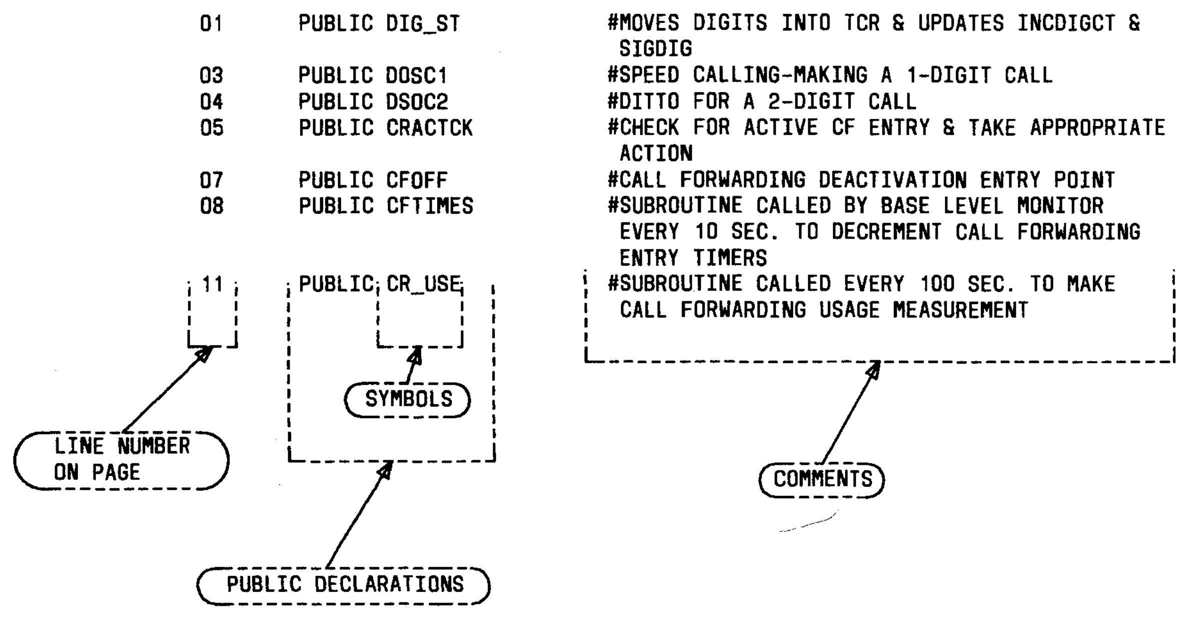
```
00054152                                    1109 02 COUNTING
00054152   1 111 D1 00                      1110 03        ST     R1,0(RA1)             # UPDATE INITQCD
00054153                                    1111 04        IF     R1 = 0 THEN RGBEGIN   # END OF INITIALIZATION INTERVAL
00054153     030 01 01                     -002- 05          TZ     R1
00054154     132 022          0054176      -001- 06          BNC    IFS553
00054155     071 01 0021024   CTSD          1112 07          STL    R1,INITLVL         # ZERO LEVEL COUNT
                                            1113 08 # SYSTEM IS OUT OF CRITICAL INTERVAL RIGHT HERE
                                            1114 09 # IE, NEXT INITIALIZATION WILL BE LEVEL 1
00054157                                    1115 10          CALL    OPPOSTMO          # TRIGGER PRINTING OF POST-MORTEM DUMP
00054157     076 01 0051073   CINIT        -001- 11          BSA     OPPOSTMO
00054161                                    1116 12          CALL    INITQEND
00054161   1 076 01 0057401   BLMMA        -001- 13          BSA     INITQEND
00054163   1 006 16 00                      1117 14          ZR      R14               # ZERO BITS 19-16 OF 20-BIT CONSTANT USED BY
                                                                                         EXCOFLMG
00054164     007 17 01 000300               1118 16          LI      R15,ES(ISC2,ISC1) # ZERO ISC BITS NOW THAT SYSTEM HAS
                                                                                          COMPLETED THE INITIALIZATION INTERVAL
00054166     007 00 01 107161               1119 18          LI      R0,SS_R%T*E(8):AR%F
                                            1120 19 # EXCOFLMG IS A SPECIAL ENTRY TO THE INIT_OCC SUBROUTINE.
                                            1121 20 # WITH R0 AND R15 SET UP AS ABOVE, THE ISC BITS WILL BE
                                            1122 21 # CLEARED IN ADDITION TO THE REST OF THE INITIALIZATION.
                                            1123 22 # OFF-LINE CANNOT BE INITIALIZED IF IT IS CURRENTLY IN USE BY A
                                            1124 23 # PROGRAM (OSA_TBLA) OR BY A PERSON (OSM_OFL).
00054170     037 02 07 003 00               1125 24          CIRM    R2,0,S80SA[TBLA),ES(OSM_OFL,OSA_TBLA)/ES(OSA_TBLA)
00054172                                    1126 25          IF      CF THEN CALL EXCOFLMG
00054172     132 003          0054175      -002- 02          BNC     IFS558
00054173     076 01 0060715   CSYSUB       -003- 27          BSA     EXCOFLMG
00054175                                   -002- 28 IFS558
00054175   1 017 17 17                      1127 29          LSR     SS_R,R15
00054176                                    1128 30          RGEND
00054176                                   -001- 31 IFS553
00054176                                    1129 32 OSCHKS

00054176   1 033 00 02                      1131 38       COM     R0,R2                # PRINT MCH FAILURE MESSAGE IF MCH WENT OUT
                                                                                         OF SERVICE SINCE LAST ENTRY
00054177     050 00 14                      1132 40       TBN     R0,S(MCH_OOS)        # CF=1 IF MCH IS STILL IN SERVICE
00054200     006 00 00        CINIT         1133 41       LN      R0,ERPMCH
00054201                                    1134 42       CALL    ERRPRTCK
00054201     076 01 0051637   CINIT        -001- 43       BSA     ERRPRTCK
00054203                                    1135 44       LMCH    RTNMB                # GET MANUAL SWITCH
00054203   1 007 00 01 000243              -001- 45       LI      R0,RTNMB
00054205                                   -001- 46       CALL    SMCH
00054205     076 01 0060551   CSYSUB       -002- 47       BSA     SMCH
00054207   1 006 03 00                      1136 48       ZR      R3                   # INIT FOR FOLLOWING HN R3,3 IN CASE RANGE
                                                                                         IS SKIPPED
00054210                                    1137 50       IF      CF THEN RGBEGIN      # USE DATA ONLY IF MCH WORKS
```

Fig. 3—Block Prologue and Comments Example



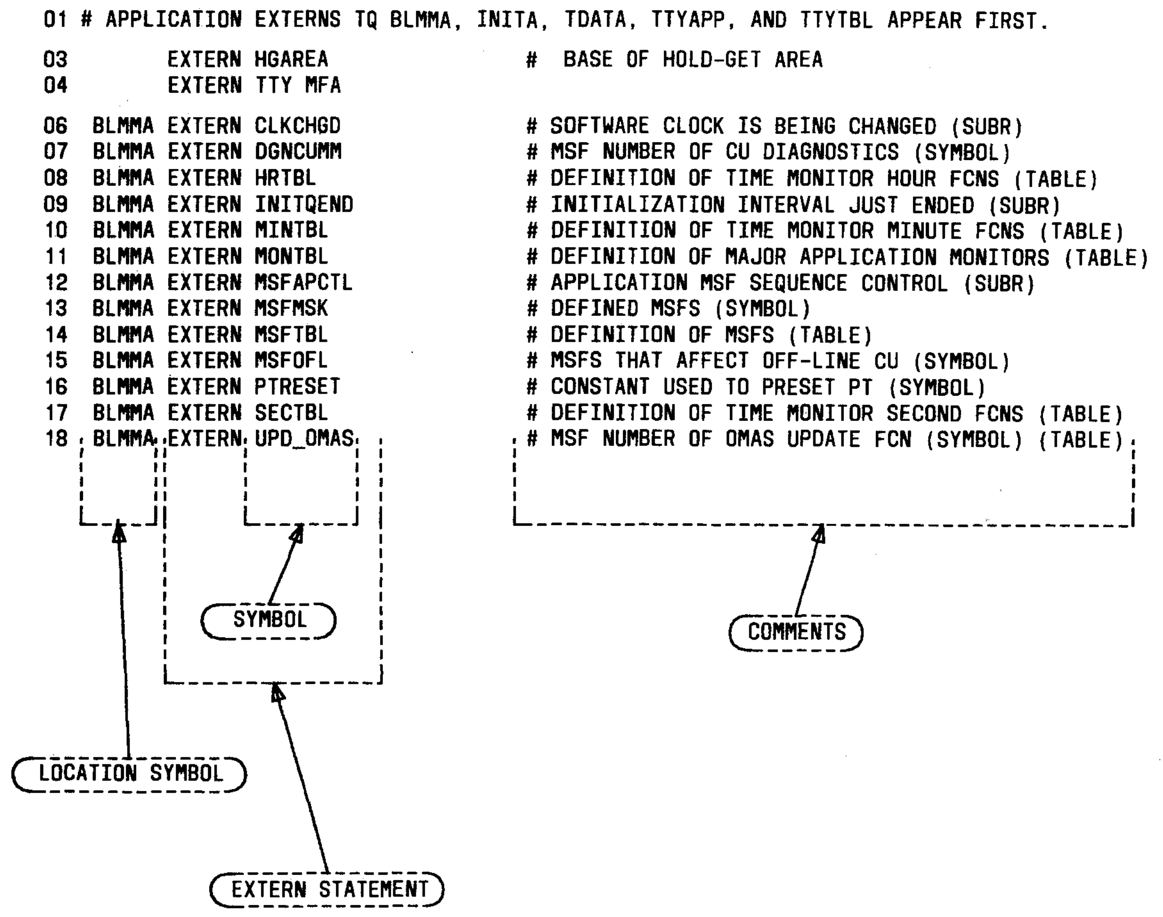
```
                                                                          CARTU 1  SYSM A022
          GENERIC 1
   OSOPS  PATCH   BB4 D9C5D7                  #PATCH FOR REP CHANNNEL IN IMOM
          GETLIB  TNS,MEMBER=TNMAC
          GETLIB  MEMBER=CCSYM
   OSTAB1 EXTERN  WPTBL                       #WRITE PROTECTION TABLE
   TNWP   EXTERN  TNSTBLP                     #TN SYSTEM TABLE
   CRTMON EXTERN  ALLOW                       #EVENT MASK FOR RESIDENT MONITOR
   CRTMON EXTERN  ETV                         #EVENT TRANSFER VECTOR FOR RESIDENT MONITOR
   CRTMON EXTERN  STATUS                      #PROGRAM STATUS BITS
   CRTMON EXTERN  MESSAGE                     #EVENT 3 (INPUT MESSAGE) ROUTINE
   AMON   EXTERN  ALLMOD                      #WORD INDICATING THAT TRANSLATION AREA
                                              #HAS BEEN MODIFIED
   CRTMON EXTERN  IDL_CRT                     #RETURN POINT TO RESIDENT MONITOR
          EXTERN  STRLIM                      #HIGHEST ADDRESS IN MEMORY
   TNWP   EXTERN  MEA                         #MASTER BLOCK ARRAY - START OF TRANSLATIONS
   TNWP   EXTERN  MEM_USED                    #HIGHEST ADDRESS USED BY FIXED TRANSLATIONS
          GETLIB  TNS,MEMBER=COMMSYMA
          GETLIB  TNS,MEMBER=COMMSYMS
          GETLIB  MEMBER=MAINTMAC
          GETLIB  MEMBER=MAINTSYM
          GETLIB  MEMBER=MAINTATT
          GETLIB  TNS,MEMBER=DSTMACM
          PUBLIC  CARTU
          NAME    CARTU
                                              #MACPRNT  OFF
             COLS OFF
             RUBOUT

             BEGIN_EPL EXPAND=OFF,PROCESSOR=AP3
                 •
                 •
                 •
                 •
                 •
                 •
              END_EPL
              END
```

Fig. 4—Control Statements Examples



```
01      PUBLIC DIG_ST          #MOVES DIGITS INTO TCR & UPDATES INCDIGCT &
                                SIGDIG
03      PUBLIC DOSC1           #SPEED CALLING-MAKING A 1-DIGIT CALL
04      PUBLIC DSOC2           #DITTO FOR A 2-DIGIT CALL
05      PUBLIC CRACTCK         #CHECK FOR ACTIVE CF ENTRY & TAKE APPROPRIATE
                                ACTION
07      PUBLIC CFOFF           #CALL FORWARDING DEACTIVATION ENTRY POINT
08      PUBLIC CFTIMES         #SUBROUTINE CALLED BY BASE LEVEL MONITOR
                                EVERY 10 SEC. TO DECREMENT CALL FORWARDING
                                ENTRY TIMERS
11      PUBLIC CR_USE          #SUBROUTINE CALLED EVERY 100 SEC. TO MAKE
                                CALL FORWARDING USAGE MEASUREMENT
```

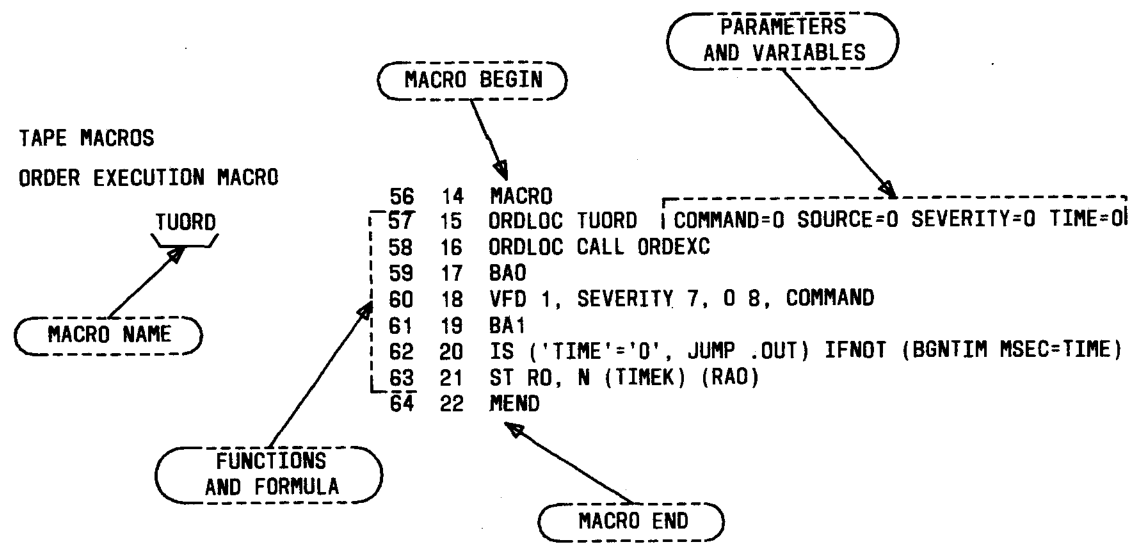
Fig. 5—PUBLIC Declaration Examples



```
# APPLICATION EXTERNS TQ BLMMA, INITA, TDATA, TTYAPP, AND TTYBTL APPEAR FIRST.
      EXTERN HGAREA              #  BASE OF HOLD-GET AREA
      EXTERN TTY MFA
BLMMA EXTERN CLKCHGD             # SOFTWARE CLOCK IS BEING CHANGED (SUBR)
BLMMA EXTERN DGNCUMM             # MSF NUMBER OF CU DIAGNOSTICS (SYMBOL)
BLMMA EXTERN HRTBL               # DEFINITION OF TIME MONITOR HOUR FCNS (TABLE)
BLMMA EXTERN INITQEND            # INITIALIZATION INTERVAL JUST ENDED (SUBR)
BLMMA EXTERN MINTBL              # DEFINITION OF TIME MONITOR MINUTE FCNS (TABLE)
BLMMA EXTERN MONTBL              # DEFINITION OF MAJOR APPLICATION MONITORS (TABLE)
BLMMA EXTERN MSFAPCTL            # APPLICATION MSF SEQUENCE CONTROL (SUBR)
BLMMA EXTERN MSFMSK              # DEFINED MSFS (SYMBOL)
BLMMA EXTERN MSFTBL              # DEFINITION OF MSFS (TABLE)
BLMMA EXTERN MSFOFL              # MSFS THAT AFFECT OFF-LINE CU (SYMBOL)
BLMMA EXTERN PTRESET             # CONSTANT USED TO PRESET PT (SYMBOL)
BLMMA EXTERN SECTBL              # DEFINITION OF TIME MONITOR SECOND FCNS (TABLE)
BLMMA EXTERN UPD_OMAS            # MSF NUMBER OF OMAS UPDATE FCN (SYMBOL) (TABLE)
```

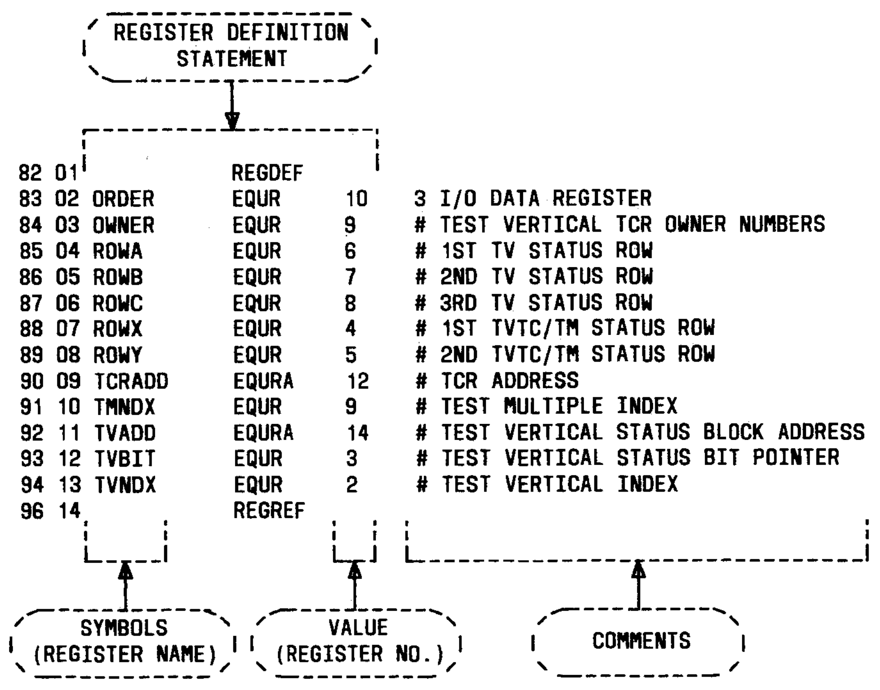
Fig. 6—EXTERN Declaration Examples

```
15:44 08  8/08.72   ****
   CTAPH  SYSM    CS31
```



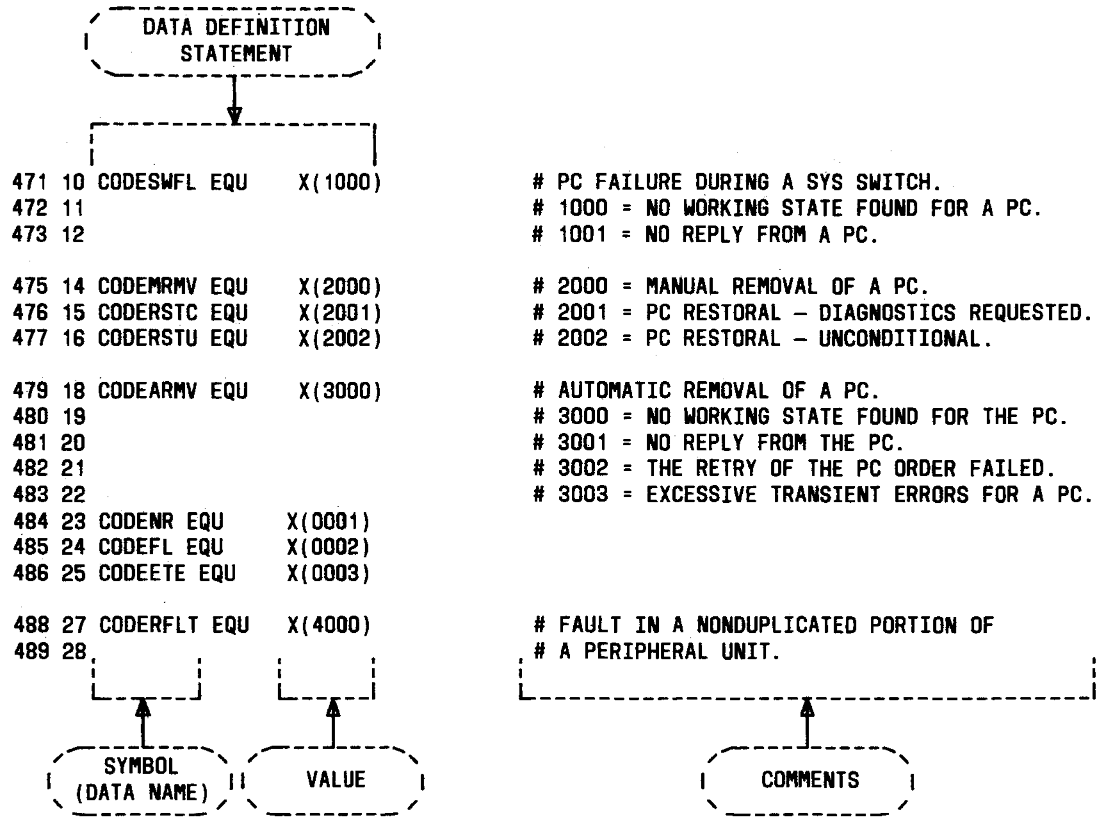
```
TAPE MACROS
ORDER EXECUTION MACRO
              MACRO
   TUORD      ORDLOC TUORD  COMMAND=0 SOURCE=0 SEVERITY=0 TIME=0
              ORDLOC CALL ORDEXC
              BAO
              VFD 1, SEVERITY 7, 0 8, COMMAND
              BA1
              IS ('TIME'='0', JUMP .OUT) IFNOT (BGNTIM MSEC=TIME)
              ST RO, N (TIMEK) (RAO)
              MEND
```

Fig. 7—Macro Definition Example



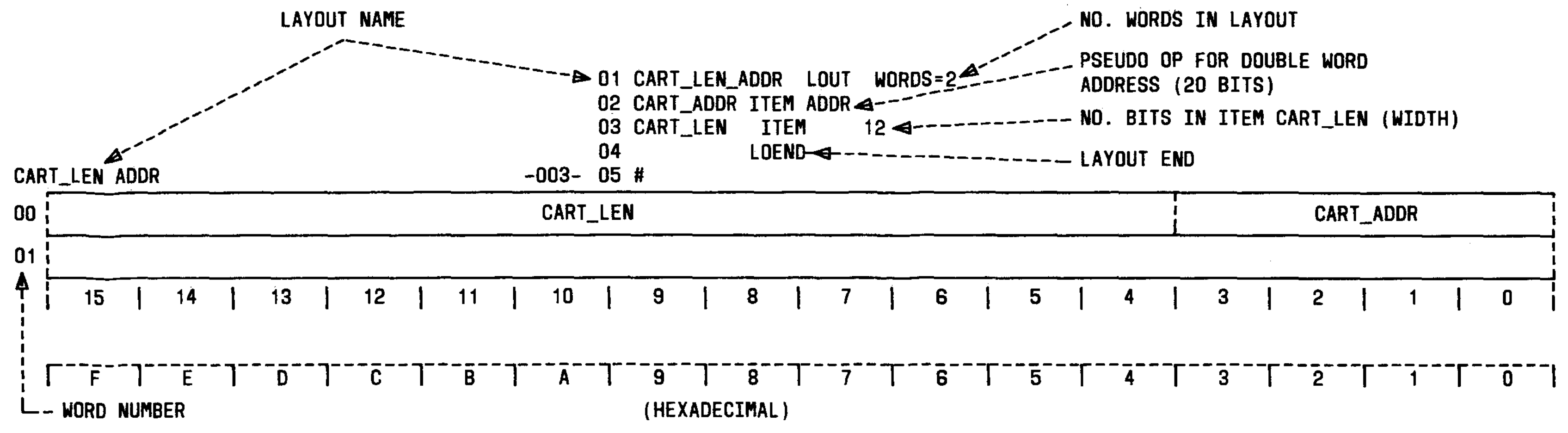
```
        REGDEF
ORDER   EQUR    10    3 I/O DATA REGISTER
OWNER   EQUR    9     # TEST VERTICAL TCR OWNER NUMBERS
ROWA    EQUR    6     # 1ST TV STATUS ROW
ROWB    EQUR    7     # 2ND TV STATUS ROW
ROWC    EQUR    8     # 3RD TV STATUS ROW
ROWX    EQUR    4     # 1ST TVTC/TM STATUS ROW
ROWY    EQUR    5     # 2ND TVTC/TM STATUS ROW
TCRADD  EQURA   12    # TCR ADDRESS
TMNDX   EQUR    9     # TEST MULTIPLE INDEX
TVADD   EQURA   14    # TEST VERTICAL STATUS BLOCK ADDRESS
TVBIT   EQUR    3     # TEST VERTICAL STATUS BIT POINTER
TVNDX   EQUR    2     # TEST VERTICAL INDEX
        REGREF
```

Fig. 8—Symbol Definition Example for Resisters

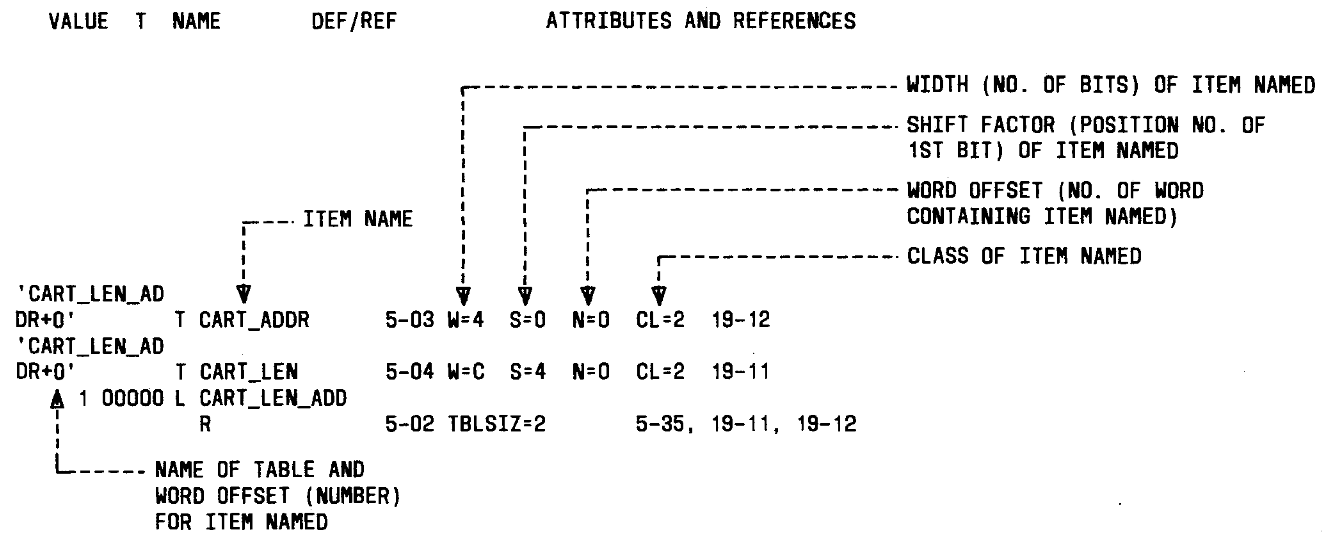


```
CODESWFL EQU    X(1000)        # PC FAILURE DURING A SYS SWITCH.
                               # 1000 = NO WORKING STATE FOUND FOR A PC.
                               # 1001 = NO REPLY FROM A PC.

CODEMRMV EQU    X(2000)        # 2000 = MANUAL REMOVAL OF A PC.
CODERSTC EQU    X(2001)        # 2001 = PC RESTORAL - DIAGNOSTICS REQUESTED.
CODERSTU EQU    X(2002)        # 2002 = PC RESTORAL - UNCONDITIONAL.

CODEARMV EQU    X(3000)        # AUTOMATIC REMOVAL OF A PC.
                               # 3000 = NO WORKING STATE FOUND FOR THE PC.
                               # 3001 = NO REPLY FROM THE PC.
                               # 3002 = THE RETRY OF THE PC ORDER FAILED.
                               # 3003 = EXCESSIVE TRANSIENT ERRORS FOR A PC.
CODENR EQU     X(0001)
CODEFL EQU     X(0002)
CODEETE EQU    X(0003)

CODERFLT EQU   X(4000)         # FAULT IN A NONDUPLICATED PORTION OF
                               # A PERIPHERAL UNIT.
```

Fig. 9—Data Definition Example



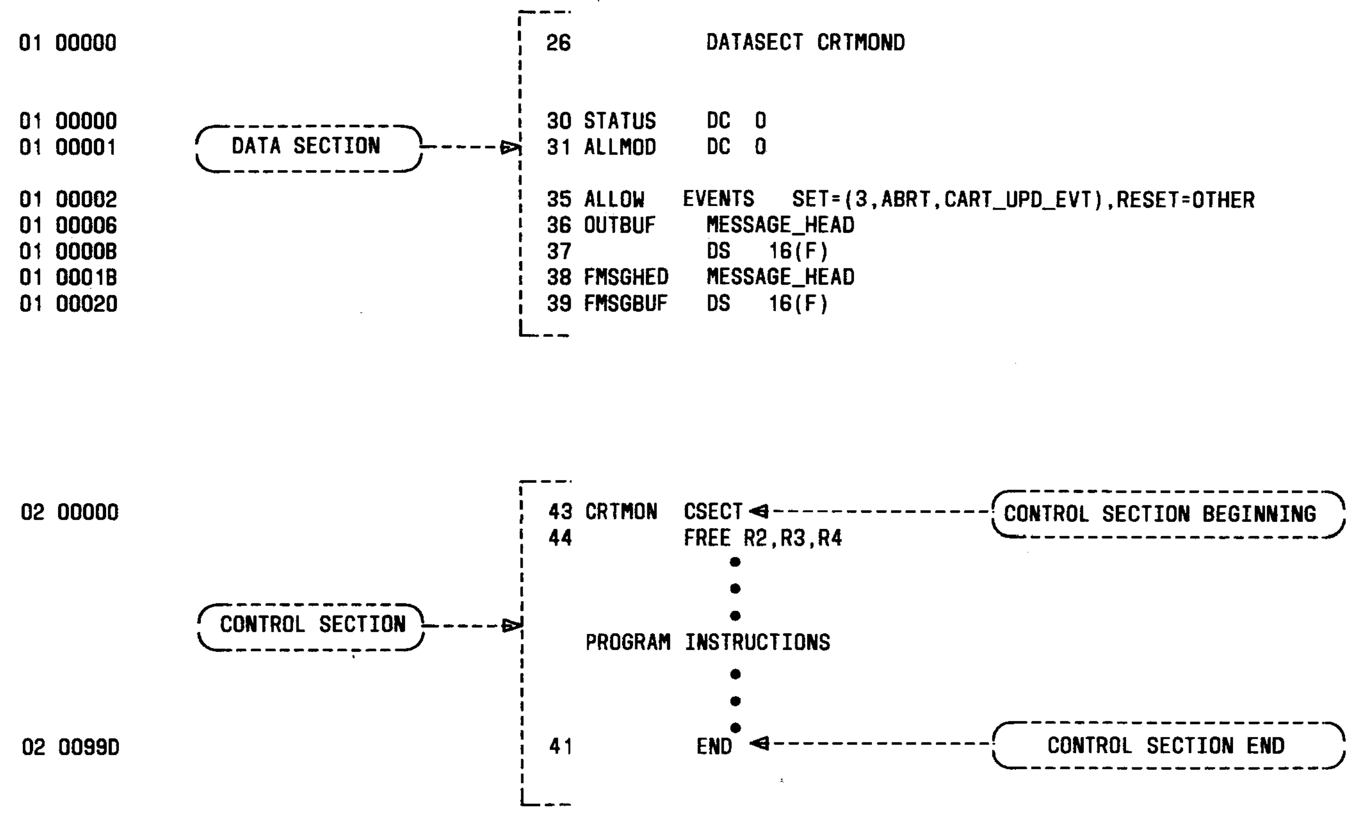
```
                                  01 CART_LEN_ADDR  LOUT  WORDS=2
                                  02 CART_ADDR ITEM ADDR
                                  03 CART_LEN   ITEM     12
                                  04              LOEND
CART_LEN ADDR               -003- 05 #
```



```
VALUE  T  NAME        DEF/REF         ATTRIBUTES AND REFERENCES

'CART_LEN_AD
DR+0'      T CART_ADDR       5-03 W=4  S=0  N=0  CL=2  19-12
'CART_LEN_AD
DR+0'      T CART_LEN        5-04 W=C  S=4  N=0  CL=2  19-11
  1 00000 L CART_LEN_ADD
             R               5-02 TBLSIZ=2       5-35, 19-11, 19-12
```

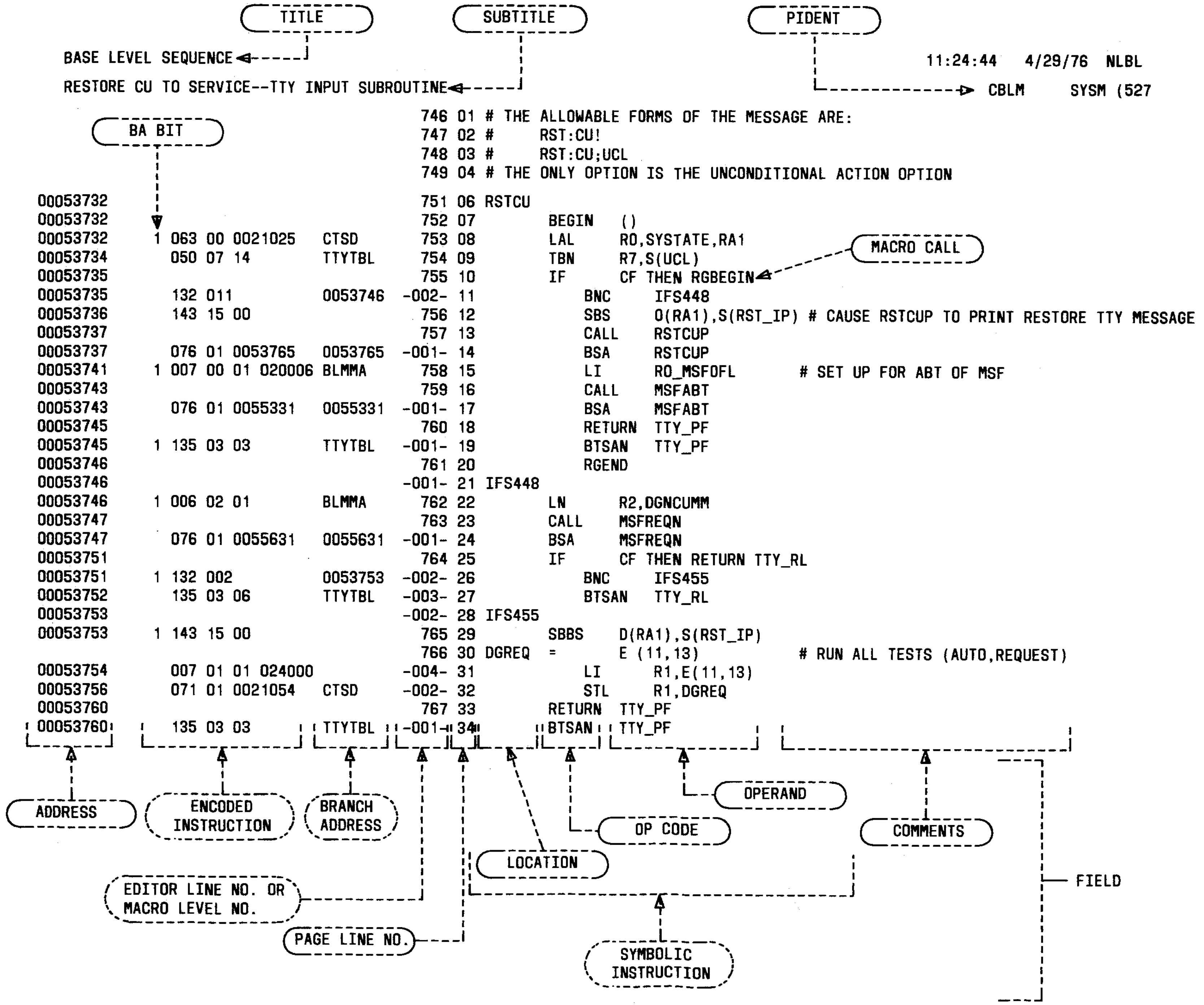
WIDTH (NO. OF BITS) OF ITEM NAMED
SHIFT FACTOR (POSITION NO. OF 1ST BIT) OF ITEM NAMED
WORD OFFSET (NO. OF WORD CONTAINING ITEM NAMED)
CLASS OF ITEM NAMED
ITEM NAME
NAME OF TABLE AND WORD OFFSET (NUMBER) FOR ITEM NAMED

Fig. 10—Memory Table Pictorial Layout and Associated Reference in the Symbols Reference Table



```
01 00000                                     26          DATASECT CRTMOND

01 00000                                     30 STATUS    DC  0
01 00001                                     31 ALLMOD    DC  0

01 00002                                     35 ALLOW   EVENTS   SET=(3,ABRT,CART_UPD_EVT),RESET=OTHER
01 00006                                     36 OUTBUF    MESSAGE_HEAD
01 0000B                                     37           DS   16(F)
01 0001B                                     38 FMSGHED   MESSAGE_HEAD
01 00020                                     39 FMSGBUF   DS   16(F)


02 00000                                     43 CRTMON  CSECT
                                             44         FREE R2,R3,R4
                                                           •
                                                           •
                                                           •
                                                PROGRAM INSTRUCTIONS
                                                           •
                                                           •
                                                           •
02 0099D                                     41         END
```

Fig. 11—DATASECT and CSECT Examples



```
BASE LEVEL SEQUENCE
RESTORE CU TO SERVICE--TTY INPUT SUBROUTINE                          11:24:44   4/29/76  NLBL
                                                                     CBLM      SYSM (527
                                         746 01 # THE ALLOWABLE FORMS OF THE MESSAGE ARE:
                                         747 02 #      RST:CU!
                                         748 03 #      RST:CU;UCL
                                         749 04 # THE ONLY OPTION IS THE UNCONDITIONAL ACTION OPTION
00053732                                 751 06 RSTCU
00053732                                 752 07         BEGIN   ()
00053732   1 063 00 0021025   CTSD       753 08         LAL     R0,SYSTATE,RA1
00053734     050 07 14        TTYTBL     754 09         TBN     R7,S(UCL)
00053735                                 755 10         IF      CF THEN RGBEGIN
00053735     132 011          0053746  -002- 11             BNC     IFS448
00053736     143 15 00                   756 12             SBS     0(RA1),S(RST_IP) # CAUSE RSTCUP TO PRINT RESTORE TTY MESSAGE
00053737                                 757 13             CALL    RSTCUP
00053737     076 01 0053765   0053765  -001- 14             BSA     RSTCUP
00053741   1 007 00 01 020006 BLMMA      758 15             LI      R0_MSFOFL       # SET UP FOR ABT OF MSF
00053743                                 759 16             CALL    MSFABT
00053743     076 01 0055331   0055331  -001- 17             BSA     MSFABT
00053745                                 760 18             RETURN  TTY_PF
00053745   1 135 03 03        TTYTBL   -001- 19             BTSAN   TTY_PF
00053746                                 761 20             RGEND
00053746                               -001- 21 IFS448
00053746   1 006 02 01        BLMMA      762 22         LN      R2,DGNCUMM
00053747                                 763 23         CALL    MSFREQN
00053747     076 01 0055631   0055631  -001- 24         BSA     MSFREQN
00053751                                 764 25         IF      CF THEN RETURN TTY_RL
00053751   1 132 002          0053753  -002- 26             BNC     IFS455
00053752     135 03 06        TTYTBL   -003- 27             BTSAN   TTY_RL
00053753                               -002- 28 IFS455
00053753   1 143 15 00                   765 29         SBBS    D(RA1),S(RST_IP)
                                         766 30 DGREQ   =       E (11,13)       # RUN ALL TESTS (AUTO,REQUEST)
00053754     007 01 01 024000          -004- 31             LI      R1,E(11,13)
00053756     071 01 0021054   CTSD     -002- 32             STL     R1,DGREQ
00053760                                 767 33         RETURN  TTY_PF
00053760     135 03 03        TTYTBL   -001- 34             BTSAN   TTY_PF
```

Fig. 12—Fields of Program Unit Listing—Typical Page Example

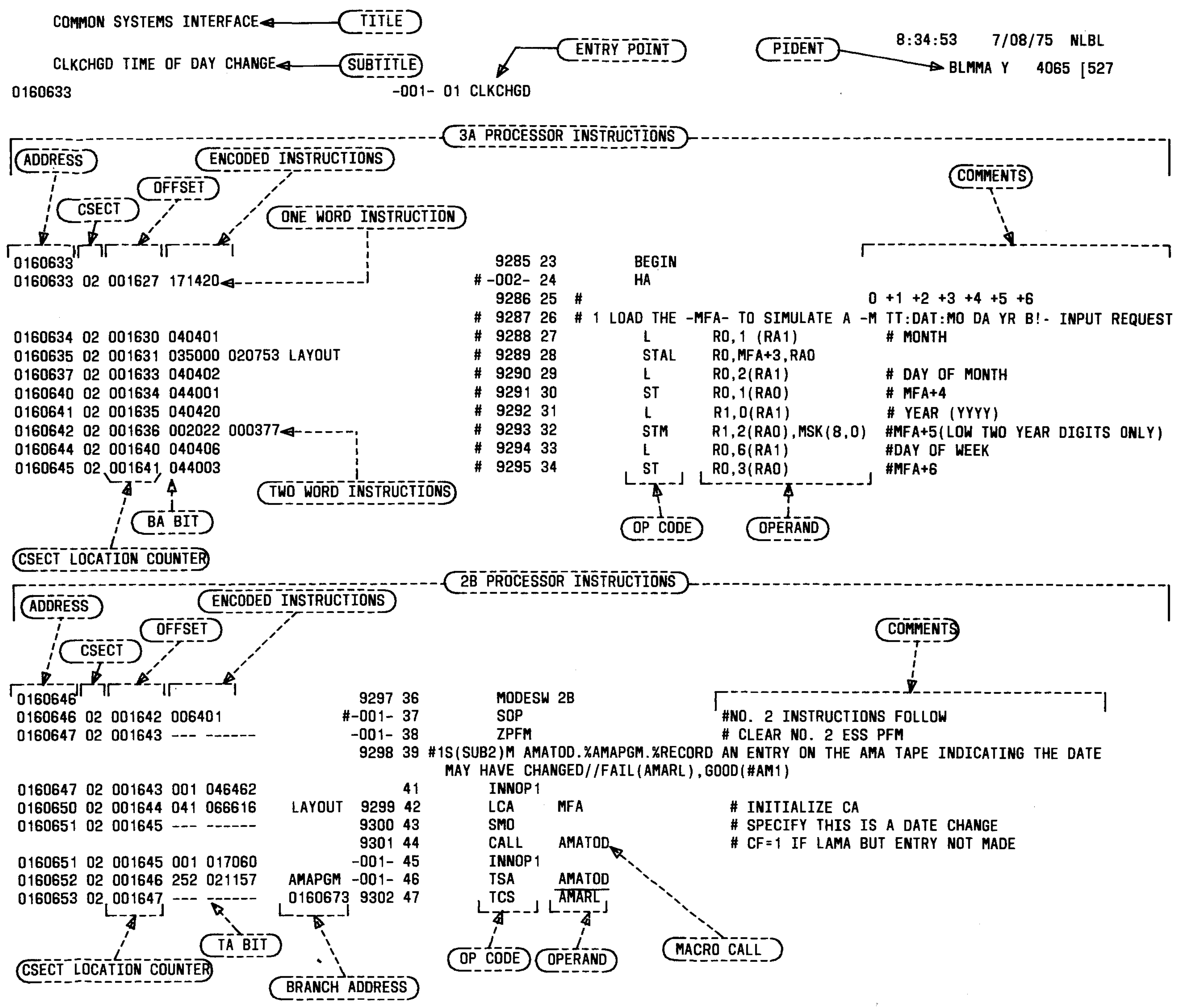
UNEXPANDED EPL LISTING

```
'P_DATE', PRINT DATE                                                              20:29:58    7/28/77  NLBL

                                                                                         CARTU 1   SYSM A022
                                               01 P_DATE      ROUTINE  SAVE=ALL
02 004E4                                       02             CMCNTL   OBUF,PACTION=M
02 004E8                                       03             OMFMT    OBUF,(WRD,WRD,WRD,WRD,DEC,ASC,ASC,ASC,ASC)
02 004F0                                       04             COMPOSE  <OBUF+4>,(V,F,Y)
02 004F4                                       05             COMPOSE  <OBUF+5>,(D,A,T,E)
02 004F8                                       06             IF TRNS_NOW<STATUS> EQ YES_ THEN COMPOSE <OBUF+6>,(C,U,R,R,E,N,T)
                                                                                       ELSE   COMPOSE <OBUF+6>,(O,L,D)
02 00505                                       08             COMPOSE <OBUF+7>,(T,D,C)
02 00509                                       09             MULT_PRINT     OBUF,CLASS=9,FREE=R6
02 0051A                                       10             S_RETURN
```

EXPANDED EPL LISTING

```
NETWORK FABRIC EXERCISE                EXPANDED INSTRUCTIONS                  16:24:14    10/08/76  NLBL

                                                                                       NFEX   SYSM A213
00011445                                394 39  LET <TCR+6> = R1
00011445     1 110 01 06               -001- 40    ST       R1.6(TCR)
00011446                                395 41                                # EXERCISE ALL SWITCHES IN THE CONCENTRATOR
                                                                                GROUP.
00011446                                396 43  LET <TCR+7> = (<TCR+8> = 0)
00011446      006 02 00                -001-:                                 # MAKE REQUEST UNCONDITONAL.
00011447      110 02 10                -001- 45    ST       R2.8(TCR)
00011450      110 02 07                -001-:
00011451                                397 47
00011451      007 00 01 010000 TTYTBL   398 48        L1        R0.M(UCL)
00011453                                399 49  LET <TCR+9> = R0
00011453      110 00 11                -001- 50    ST       R0.9(TCR)
```

Fig. 13—Unexpanded and Expanded EPL Listing—Typical Page Examples



```
COMMON SYSTEMS INTERFACE
                                                                        8:34:53    7/08/75  NLBL
CLKCHGD TIME OF DAY CHANGE                                              BLMMA Y    4065 [527
0160633                                       -001- 01 CLKCHGD

0160633
0160633 02 001627 171420                           9285 23        BEGIN
                                                # -002- 24        HA
                                                   9286 25  #                            0 +1 +2 +3 +4 +5 +6
                                                #  9287 26  # 1 LOAD THE -MFA- TO SIMULATE A -M TT:DAT:MO DA YR B!- INPUT REQUEST
0160634 02 001630 040401                        #  9288 27         L      RO,1 (RA1)          # MONTH
0160635 02 001631 035000 020753 LAYOUT          #  9289 28         STAL   RO,MFA+3,RAO
0160637 02 001633 040402                        #  9290 29         L      RO,2(RA1)           # DAY OF MONTH
0160640 02 001634 044001                        #  9291 30         ST     RO,1(RAO)           # MFA+4
0160641 02 001635 040420                        #  9292 31         L      R1,0(RA1)           # YEAR (YYYY)
0160642 02 001636 002022 000377                 #  9293 32         STM    R1,2(RAO),MSK(8,0)  #MFA+5(LOW TWO YEAR DIGITS ONLY)
0160644 02 001640 040406                        #  9294 33         L      RO,6(RA1)           #DAY OF WEEK
0160645 02 001641 044003                        #  9295 34         ST     RO,3(RAO)           #MFA+6
```

```
0160646                               9297 36        MODESW 2B
0160646 02 001642 006401             #-001- 37        SOP                             #NO. 2 INSTRUCTIONS FOLLOW
0160647 02 001643 --- ------          -001- 38        ZPFM                            # CLEAR NO. 2 ESS PFM
                                      9298 39 #1S(SUB2)M AMATOD.%AMAPGM.%RECORD AN ENTRY ON THE AMA TAPE INDICATING THE DATE
                                         MAY HAVE CHANGED//FAIL(AMARL),GOOD(#AM1)
0160647 02 001643 001 046462             41        INNOP1
0160650 02 001644 041 066616  LAYOUT  9299 42        LCA      MFA                    # INITIALIZE CA
0160651 02 001645 --- ------          9300 43        SMO                             # SPECIFY THIS IS A DATE CHANGE
                                      9301 44        CALL     AMATOD                 # CF=1 IF LAMA BUT ENTRY NOT MADE
0160651 02 001645 001 017060          -001- 45        INNOP1
0160652 02 001646 252 021157  AMAPGM -001- 46        TSA      AMATOD
0160653 02 001647 --- ------   0160673 9302 47        TCS      AMARL
```

Fig. 14—Program Unit Page Listing—Typical Page Examples

```
----------------------------EPILOGUE --------------------------    14:02:05    4/10/75  NLBL
                                                                        TVADM      SYSM 123
      VALUE T NAME       DEF/REF        ATTRIBUTES AND REFERENCES
                                     38-26, 38-42, 39-16, 39-25, 39-31, 39-32, 39-38
          9 A ROWX%SHIFT  32-14
          5 R ROWY         9-09  28-24, 28-26, 28-28, 32-05, 38-11, 38-24
'TV_STAT+12" T RVGS_QC    22-02 W=2  S=2  N=C  CL=2
'TV_STAT+12" T RVLP_QC    22-03 W=2  S=4  N=C  CL=2
          0 A RVRS        16-17 W=1  S=F  N=0  CL=3
          6 A RVRSA       11-26 W=1  S=F  N=6  CL=3   41-19, 41-20
          8 A RVRSB       11-32 W=1  S=F  N=8  CL=3   13-23
          9 A RVRST       11-35 W=1  S=F  N=9  CL=3   13-29
          0 R RO                 29-11, 29-12, 29-14, 29-20, 29-41, 32-33, 32-34, 32-36, 32-39, 32-41, 32-42, 32-45, 32-46,
                                 33-01, 35-37, 35-43, 36-04, 36-06, 36-10, 36-12, 36-13, 36-15, 36-23, 36-35, 37-06, 38-02,
                                 38-05, 38-06, 38-38, 39-28, 41-06, 41-07, 41-08, 43-04
          D A RO%SHIFT    29-12
          1 R R1                 28-04, 28-05, 29-04, 29-08, 29-14, 29-19, 29-20, 29-21, 29-36, 29-37, 29-39, 32-35, 32-36,
                                 32-37, 32-39, 32-42, 33-05, 33-06, 35-38, 35-43, 35-46, 36-03, 36-04, 36-13, 36-14, 36-17,
                                 36-18, 36-20, 36-34, 36-45, 36-47, 37-07, 37-08, 38-29, 38-33, 38-45, 38-46, 39-08, 39-12,
                                 39-29, 39-31, 41-03, 41-11, 41-13, 41-16, 41-23, 41-28, 43-32
          4 A R1%SHIFT    33-05
          A R R10                29-15, 29-16, 29-18, 29-19, 35-11, 35-18, 35-30, 35-35, 35-48, 36-03, 36-12, 36-14, 36-15,
                                 36-30, 36-37
          4 A R10%SHIFT   29-18
          B R R11                36-22, 36-23, 36-34, 36-35, 36-36, 36-45
          9 R R9                 35-10, 35-20, 36-29, 36-38, 36-48
```

Fig. 15—Symbols Reference Table—Typical Page Example

TABLE A

ABBREVIATIONS AND ACRONYMS

ABBREVIATION/ACRONYM	TERM
ADDR	Address
AU	Assembly Unit
BA	Branch Allowed
CC	Central Control
CL	Class
COM	Complement
CSECT	Control Section
DATASECT	Data Section
DEF	Definition
EPL	ESS Programming Language
EQU	Equates
ESS	Electronic Switching System
LIB	Library
LR	Left Rotate
N	Index of Item
OP	Operation
PIDENT	Program Identification
PR	Program Listing
PU	Program Unit
R	Register
RA	Address Register
REF	Reference
S	Shift Factor (1st bit position of item)
T	Type of Symbol
TBLSIZ	Table Size (number of words)
W	Width of Item (number of bits)

TABLE B

NUMBER SYSTEMS

BINARY	OCTAL	HEXADECIMAL	DECIMAL
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	A	10
1011	13	B	11
1100	14	C	12
1101	15	D	13
1110	16	E	14
1111	17	F	15

TABLE C

TYPE SYMBOLS FROM SYMBOL REFERENCE LIST

LETTER	TYPE OF SYMBOL
A	Absolute
B	Attribute Name
D	Program Store Data
J	Truth Valued
K	Function Name
L	Program Store Location
M	Macro
N	Pure Number
O	Machine Op
P	Address Register Pair
R	Register
S	Sequence
T	Test
U	Undefined
V	EXTERN
X	DIRECT
Y	Array Name
Z	Pseudo-operation