Computer Systems Department

BUIC III PROGRAMMER COMPILER LANGUAGE TEXT

April 1968

Keesler Technical Training Center
Keesler Air Force Base, Mississippi

Designed For ATC Course Use

ATC Keesler 9- 4102

ABOUT THE STUDENT TEXT

STUDENT TEXTS are authorized by the Air Training Command as student training publications for use in training situations peculiar to courses in this Command. They contain specific information required by the student to achieve the learning objectives. It contains the necessary information which is not suitable for student study in other available publications.

STUDENT TEXTS are designed for ATC COURSE USE ONLY. Every effort is made to keep in-use student texts current with technical orders and other directives. Students are cautioned not to use them in preference to technical orders or other authorative documents. When students are authorized to retain student texts they must keep in mind that these publications will not remain current.

BUIC III PROGRAMMER COMPILER LANGUAGE TEXT

TABLE OF CONTENTS

INTRODUCTION TO JOVIAL

JOVIAL is a language with which one can write programs to solve problems on a computer. A PROGRAM is a sequence of instructions to a computer. A PROGRAM contains the logic which is designed to sove a particular problem.

Each type of computer is electrically designed to be sensitive to a specific set of computer instructions. These are usually coded in the memory of the computer in what is called the BINARY language. This is a language consisting of combinations of 0's and 1's. The binary language is called a LOW LEVEL LANGUAGE.

Usually each type of computer has a set of symbolic instructions corresponding to the set of binary instructions. These are coded using alphabetic abbreviations and are called SYMBOLIC MACHINE-INSTRUCTIONS. This symbolic machine language is called an INTERMEDIATE LEVEL LANGUAGE.

A disadvantage of symbolic machine language is that it is different for each make of computer. Furthermore, a programmer using symbolic machine language has to pay close attention to details peculiar to his machine. This detracts from his efforts to form the logic connected with the solution of the problem he is programming.

To overcome the aforementioned disadvantages of intermediate level languages, HIGH LEVEL LANGUAGES have been developed.

A high level programming language is one which more closely approaches the language of English and mathematics. It permits the programmer to be less concerned about the individual peculiarities of the specific computer involved and enables him to concentrate more conveniently on the logic involved in the solution of a problem.

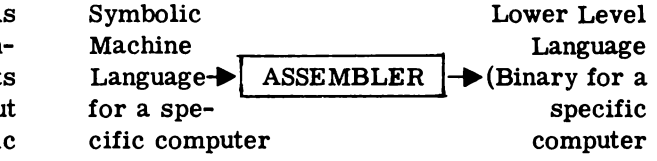
JOVIAL is a high level language. Other higher level languages are ALGOL, MAD, FORTRAN, COBOL, NELLIAC, etc. JOVIAL was developed by the System Development Corporation to be used for programming large scale command and control systems. Work on the language began in 1959. Since that time, many modifications and improvements have been made to the language.

The name JOVIAL is an acronym which is derived from Jules Own Version of the International Algebraic Language. Jules Schwartz of the System Development Corporation was the scientist in charge of the initial language development. Several colleagues of his supplied the name one time when he was away on a business trip and the name stuck. JOVIAL it is.

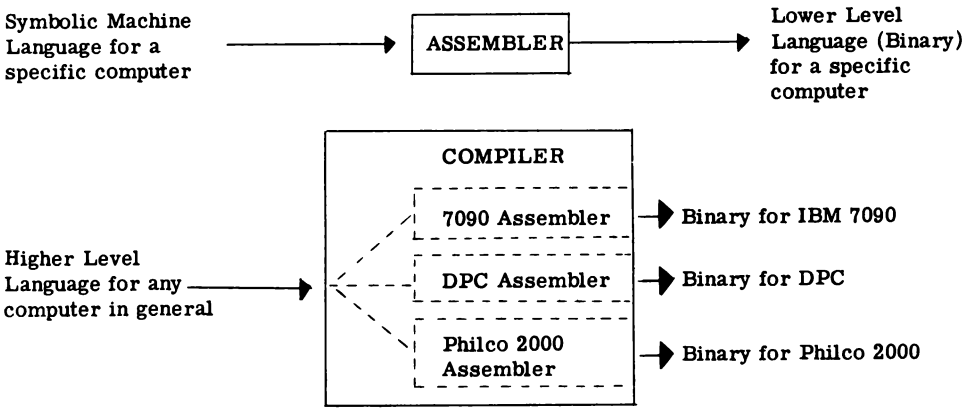
A computer program written in symbolic machine language must be translated to binary since the computer operates using instructions coded in binary. A computer program, called an ASSEMBLER, is used to perform the translation from the intermediate level language to the lower level language.

Similarly a computer program written in a higher level language must be translated to the binary language of the specific machine involved. The computer program which performs this translation is called a COMPILER.

JOVIAL compilers are in existence for several makes of computers, two of the most well-known being the IBM 7090 computer, and the AN/FSQ-31V or DPC (Data Processing Central).



The diagram shown on the next page is representative of the functions of assemblers and compilers. Usually a compiler contains only one assembler. For example, a JOVIAL compiler which translates JOVIAL program statements to IBM 7090 binary



instructions, would contain in it only a FAP assembler (the 7090 assembler).

It is probably apparent that functions in addition to assembly are performed by a compiler. This is because the compiler has to translate the higher level language into the symbolic machine language of the assembler involved. Also each statement or instruction in a higher level language is often translated into many symbolic machine instructions.

The above diagram also illustrates that a program written in a higher level language can be operated on different computers by having the translation to binary performed using the appropriate compiler. This is a distinct advantage over writing at the intermediate language level, where the program can be operated ONLY on the computer for which it was written.

ELEMENTS OF THE LANGUAGE

A program written in JOVIAL consists basically, of statements and declarations. The statements specify the computations to be performed with arbitrarily named data. There are both simple statements and complex statements which can be grouped together into compound statements. Among the declarations are data declarations and processing declarations. The data declarations name and describe the data on which the program is to operate, including inputs, intermediate results, and final results. The processing declarations generally contain statements and other declarations. They specify computations, but they differ from statements in that the computations must be performed only when the particular processing declaration is specifically invoked by name. In addition to statements and declarations there are directives by means of which the compiler is caused to change its interpretation of certain structures in the program. The statements, declarations, and directives are composed of symbols, which are the words of the JOVIAL language. These symbols are in turn composed of signs which comprise the JOVIAL alphabet.

1. SIGNS, ELEMENTS OF THE JOVIAL ALPHABET

SIGN means a letter, a numeral, or a mark.

LETTER means one of the twenty-six letters of the English alphabet, written in the form of a roman capital.

NUMERAL means one of the ten Arabic numerals Ø, 1, 2, 3, 4, 5, 6, 7, 8, 9.

OCTAL NUMERAL means one of the following numerals: Ø, 1, 2, 3, 4, 5, 6, 7.

MARK means one of the twelve marks, each associated with a name or names in parentheses, in the following list:

(1) +	(5) (blank)	(9) (
(2) -	(6) .	(10))
(3) #	(7) ,	(11) '
(4) /	(8) =	(12) $

2. SYMBOLS, THE WORDS OF JOVIAL

The symbols or words of the JOVIAL language are composed of strings of signs, in some cases a single sign. Most symbols do not contain spaces. In fact, spaces serve to separate symbols from one another. In the definitions of symbols, the phrase "enclosed in parentheses," means having a left parenthesis on the left and a right parenthesis on the right without any intervening spaces. If the input medium is cards (or card-images or other "unit records") each symbol must be completely contained on one card.

SYMBOL means one of the following expressions:

(1) primitive
(2) constant
(3) loop variable
(4) abbreviation
(5) name
(6) ideogram
(7) comment

The above definition contains a categorical listing of all JOVIAL symbols, but primitive and ideogram have reference to the way these symbols are written rather than their use in constructing programs. These two categories can be regrouped in ways that are more suggestive of their roles in the language.

Those symbols which are primitives or ideograms include the categories in the following list, which is not exhaustive:

(1) arithmetic operator
(2) relational operator
(3) logical operator
(4) functional modifier
(5) bracket

3. PRIMITIVE, NAME, LOOP VARIABLE, ABBREVIATION, IDEOGRAM, COMMENT

The following list includes all the primitives of the JOVIAL language:

ABS	ENT	'LOC	'PROGRAM
ALL	ENTRY	LQ	RETURN
AND	EQ	LS	START
ASSIGN	FOR	NENT	STOP
BEGIN	GOTO	NOT	SWITCH
BIT	GQ	NQ	TABLE
BYTE	GR	NWDSEN	TERM
CLOSE	IF	OR	TEST
DEFINE	IFEITH	ORIF	
DIRECT	ITEM	OVERLAY	
END	JOVIAL	PROC	

A primitive is a symbol consisting, usually, of two or more letters and having a specific meaning in the JOVIAL language. In the above list there are two primitives that begin with the prime. This is in accordance with a policy of requiring the spelling of any new primitive added to the language to begin with this mark. The purpose is to avoid outlawing any previously written programs by preventing the possibility of a new primitive being identical to any name.

A NAME is a string of two through EIGHT LETTERS, numberals, and primes with the following characteristics:

(1) It has no more signs than will fit in a machine word.
(2) It is not identical to any primitive.
(3) It begins with (the leftmost sign is) a letter.
(4) The rightmost sign is not a prime.
(5) It does not contain two consecutive primes.
(6) It is not identical to any of the words in the following list:

	FILE	MODE	OUTPUT
ARRAY	INPUT	ODD	POS
CHAR	MANT	OPEN	SHUT
			STRING

The above twelve words are primitives in the full JOVIAL language, but not in this subset. Nevertheless, even in the subset, names must avoid these conflicts.

LOOP VARIABLE. Any single letter can be used as a loop variable. It is the context in which it is used that characterizes it as a loop variable. A loop variable is often called by other terms such as for-variable or single-letter subscript.

ABBREVIATION. Several letters are used, standing alone, as abbreviations. The meaning of an abbreviation depends on context. Those letters which may be used as abbreviations will not be exhibited here, but will be shown and explained in connection with the forms in which they can occur.

IDEOGRAM means a string of marks having meaning in JOVIAL. Each of the twelve marks except the space and the prime is also an ideogram. Following are listed the 18 ideograms.

+	,	''	(/
-	=	($	/)
*	(	$)	**
/	)	(*	
.	$	*)	

COMMENT means two primes followed by a string of signs followed by two primes. The string of signs between the two sets of doubled primes may contain spaces. It must not contain two primes in succession; the last sign before the second set of two primes must not be a prime; and the string of signs must not contain $ except in the following two combinations:

($
$)

4. JOVIAL FORMAT

JOVIAL programs are compiled from a deck of punched cards either pre-stored on magnetic tape or read in via the on-line cardreader. Each JOVIAL program must be bracketed by two cards: the START card at the beginning of the program; and the TERM$ card at the end of the program.

Each JOVIAL statement must be followed by a $. Blanks are used to separate words in a JOVIAL statement unless a unique character separates the words. For example:

XX = AA + BB $

may be written with spaces, as above, or without spaces. The deliminaters =, +, and $ are unique and serve to separate XX, AA, and BB.

More than one JOVIAL statement may be written on a card and a JOVIAL statement may continue from one card to another. The end of a card has no significance in JOVIAL except for direct coding. The first 64 columns of a card image are normally the only ones interpreted by the compiler, unless otherwise directed by the control card.

An item may be declared at any point in a JOVIAL program. However, an item which is not a simple signed integer item must be declared prior to its use in the program; simple signed integer items need not be declared because the compiler automatically treats all undeclared simple items as 48 bit signed itegers. An item declaration has the following general format:

ITEM item name item description $

There are three types of items:

1. Arithmetic items:

   a. Integer
   b. Floating Point
   c. Fixed Point

2. Literal items:

   a. Hollerith
   b. Standard Transmission Code

3. Status items

A simple item declaration reserves a full computer word for the item. No item can exceed one computer word in length.

1. ARITHMETIC ITEM

A U or S in an arithmetic item description indicates whether the item is unsigned or signed. If the item is signed, the sign bit is included in the number of bits specified.

(1) INTEGER

I number of bits U or S

For example:

ITEM DAY I 3 U $

(2) FLOATING POINT

A floating point item always occupies a full computer word.

F

For example:

ITEM SPEED F $

(3) FIXED POINT

A number of bits U or S number of fractional bits.

For example:

ITEM XPOS A 16 S 5 $

2. LITERAL ITEM

Each character in a literal item will occupy 6 bits of a computer word.

(1) HOLLERITH

H number of characters

For example:

ITEM DAY H 3 $

(2) STANDARD TRANSMISSION CODE

T number of characters

For example:

ITEM NAME T 7 $

3. STATUS ITEM

S number of bits V(status identifier) V(status identifier) V(status identifier) etc

For example:

ITEM IDENT V(FRND) V(HOST) V(UNK) V(INT) V(FAKER) $

Note that the number of bits need not be specified. In the example cited, five statuses of identification have been specified. This indicates a range of values which requires three bits of a computer word. Each status specified is assigned a numerical value beginning with zero for the first status, one for the second, etc., up to the limit of a computer word.

4. COMPOOL DECLARATIONS

Items of any type which are used by more than one computer program may be declared in a compool and referred to by computer programs using the compool. The use of a compool removes the necessity for the repetition of item declarations by a number of computer programs when more than one computer program uses the same item at the same location.

The Control Card provides the means by which the use of a compool is indicated and the particular compool desired is specified. If the same item name is defined both by the compool and by the computer program using the compool, the computer program's definition is used.

Items defined in a compool are used by a computer program calling for the compool exactly as if the item had been declared by the computer program. All tabular items must be declared either in the compool or in the computer program before being used, or else an error will be diagnosed; all simple items common to more than one computer program must be similarly declared to establish communality; all simple items which are declared neither in the compool nor in the computer program will be treated as 48 bit signed integer items.

5. PRESETTING

The value of a simple item may be preset at the time the item is declared. This may be done in two ways:

(1) Add the preset value following the item description.

For example:

```
ITEM    FIX    A    24    S    5    P    12.5A5    $
```

In this case, the letter P indicated that the following value is initially assigned to item FIX.

(2) Use the preset value to both describe and preset the item.

For example:

```
ITEM    YPOS    135.5A5    $
```

In this case, item YPOS will be established as a fixed point item with 5 fractional bits and with a total number of bits sufficient to contain the preset value 135.5. However, caution should be exercised in presetting items in this way in case the preset value is smaller than the largest value the item is expected to attain.

Some additional examples of this method are:

```
ITEM    NAME    4H(EARL)    $
ITEM    XPOS    135.5    $
```

6. OVERLAY OF SIMPLE ITEMS

It is possible to specify that storage for simple items be allocated in particular sequences. This would not be useful except that it is also possible to specify that these sequences start in the same machine word. Thus an item may have more than one name, each name corresponding to an entirely different description of the item.

For example:

```
OVERLAY    TEMP1 = TEMP2 = TEMP3    $
```

where

TEMP1 could be treated as an integer variable
TEMP2 could be a floating variable
TEMP3 could be a literal variable

```
OVERLAY    XX,    YY,    ZZ    $
```

where

XX, YY and ZZ will be assigned consecutive storage locations.

A particular item name may be used no more than once in overlay declarations. Any item named in an overlay declaration must have been previously defined in the program or by the compool.

A table is a group of items organized as entries. The first entry of a table is defined as entry Ø; that is, when a tabular item is subscripted with a value of zero, the first value for that item in the table is indicated.

ENT and ENTRY are equivalent JOVIAL primitatives and may be used interchangeably to refer to all items of an entry of a table.

For example:

```
ENTRY   (TAB1 . ($  Ø  $))   =   Ø   $
                    or
IF  ENT  (DATA   ($INDEX$))   GR  Ø   $
```

1. DECLARATION

A table may be declared at any point in a program. However, a table must be declared prior to its use in the program. The items which make up a table must be declared as part of the table declaration. A table must have at least one item associated with it.

TABLES

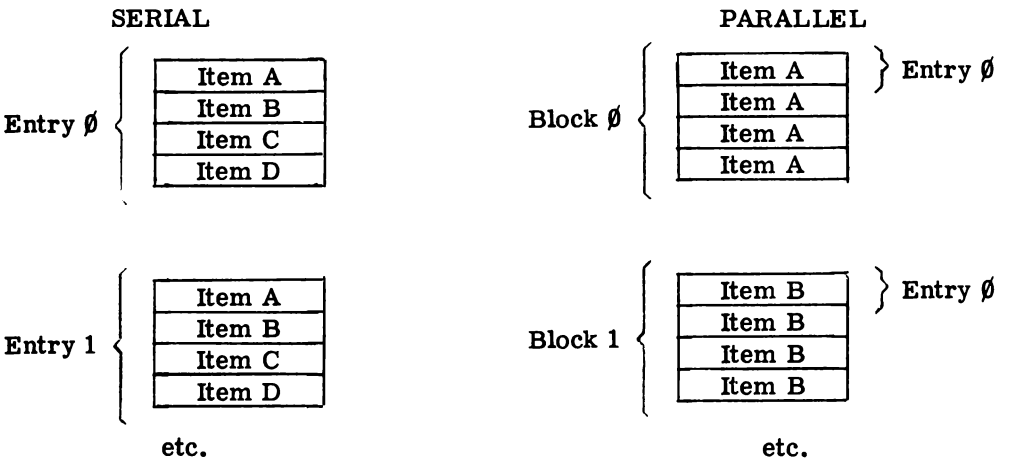
A table declaration has the following format:

TABLE table name R or V number of entries S or P N or M or D $

Table name is optional. R or V indicates whether the table is of rigid length or variable length.

The number of entries is obtained by determining the number of times the most frequently used item will appear in the table. For variable length tables, the maximum values of these numbers should be used.

S or P is optional and indicates that the table format is serial or parallel. For example:



If S or P is not specified, parallel format will be used.

N or M or D is optional and specifies the table packing. N means no packing, i.e., each item in the table will occupy a full computer word. M means medium packing. The compiler will make optimum use of 6-bit characters in assigning storage to items. D means dense packing. Each item in the table will immediately follow the previous item with no bits unused if possible. A single item will not be divided between two words.

If no packing is specified, the compiler will assume an N.

2. TABULAR ITEMS

The items to be included in a table must be declared immediately following the table and must be preceded and followed by "brackets" formed by the JOVIAL primitives BEGIN and END. An example of a complete table declaration is as follows:

```
TABLE    TRACK    V    5Ø    S    M    $
    BEGIN
        ITEM    XPOS    A    16    S    5    $
        ITEM    YPOS    A    16    S    5    $
        ITEM    IDENT    S    V(FRND)    V(HOST)    V(UNK)    $
    END
```

3. LIKE TABLES

Like table declarations are used to declare a table which is structurally similar to a previously declared table, without going through a lengthy table declaration. For example:

```
TABLE    TRACKA    L    $
```

This causes the JOVIAL compiler to reserve separate space for a table identical to the one referenced. The table description and the items within the like table will hve the same format as the referenced table.

The like table declaration must follow the declaration of the referenced table and must use the same table name plus any alphabetic character suffixed to the name.

When referring to items within a like table, the suffix of the table must be added to the item name. Like table declarations may also have a different format from the previously declared table. For example, to set up a single entry table with the same item configuration as the table named TRACK, the following like table declaration may be used:

```
TABLE    TRACKA    R    1    L    $
```

A like table may also have different structure and packing from the original table by including appropriate variations in the like table declaration.

4. OVERLAY TABLES

Tables may be assigned to share common storage space by use of the OVERLAY statement. The tables must have been declared prior to their appearance in an OVERLAY statement. For example:

```
OVERLAY    TAB1    =    TAB2    $
```

This will cause the tables TAB1 and TAB2 to begin at the same storage location. It is the programmer's responsibility to determine which table's data is occupying the shared storage space at any given time. The OVERLAY statement may also be used to cause tables to occupy consecutive locations in the computer's storage space. For example:

```
OVERLAY    TAB1,    TAB2    $
```

will cause table TAB2 to follow TAB1 immediately.

The two uses of the overlay statement may appear in combination:

```
OVERLAY    TAB1,    TAB2    =    TAB3    $
```

This will cause tables TAB1 and TAB2 to occupy consecutive locations in storage, and TAB3 will begin at the same storage location as TAB1.

If the overlay declaration contains a number or an octal constant, the common origin of the sequences will be the location identified by the value of the constant. For example:

```
OVERLAY    1024    =    TAB1    $
OVERLAY    0(1000)    =    TAB2    $
```

The name of a data structure may appear no more than once in overlay declarations. Data structures named in an overlay must first be defined, either by COMPOOL or declaration in the program. COMPOOL-defined tables in the overlay declaration must precede all other names. Overlayed tables must not be provided with preset values.

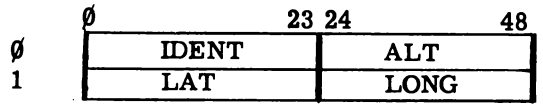
5. PROGRAMMER-DESIGNATED PACKING

The packing of a table may be designated by the JOVIAL programmer as part of the table declaration. It will not be possible to preset the values of items in such tables.

Example of designated packing:

```
TABLE    UFO    V    5ØØ    S    $

   BEGIN
      ITEM    IDENT    H    4    Ø    Ø    $
      ITEM    ALT      I    24   Ø    24   $
      ITEM    LAT      I    24   U    1    Ø    $

      ITEM    LONG     I    24   U    1    24   $
   END
```

The Ø Ø following the description of item IDENT indicates that the item will be stored in the first word of each entry and will begin with the first bit of the word. For item ALT, the item will be stored in the first word of each entry and will start at bit 24. Items LAT and LONG will be stored in word 1, or the second word, of each entry and will start at bits Ø and 24, respectively.



It will also be possible to overlay items in tables packed in this manner. For example, if the following item were added to the table UFO:

```
ITEM    SOURCE    I    28    U    1    Ø    $
```

This item would overlay the items LAT and LONG in word 1 of each entry.

6. COMPOOL DECLARATION

The same rules for compool declaration of items apply to tables.

7. PRESETTING

Items within a table are preset by specifying the preset values between BEGIN...END brackets immediately following the item declaration. The first value specified will be assigned to the item in entry Ø, the second value to the item in entry 1, etc., for as many preset values as are specified. The maximum number of preset values is the same as the number of entries for the item. For example:

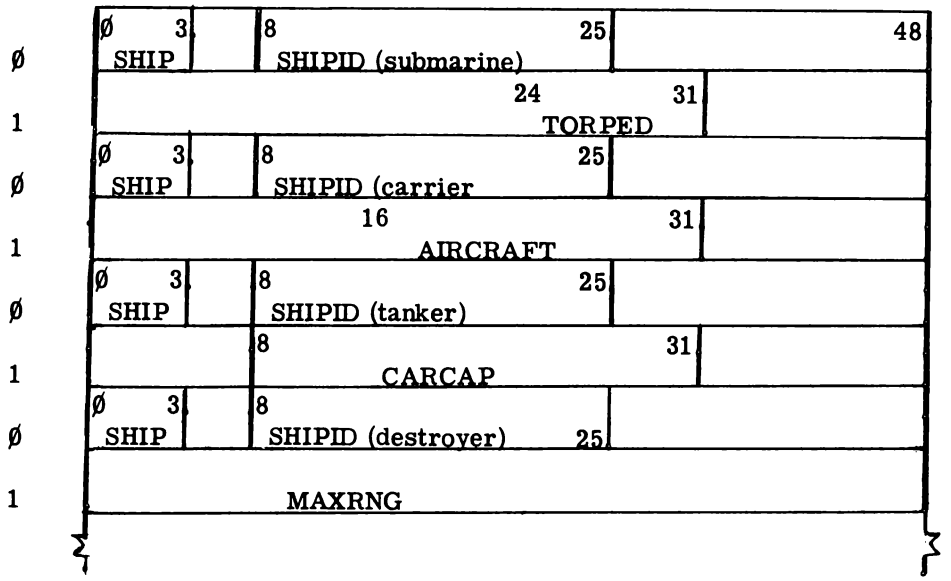
```
TABLE    R    5Ø    M    $
    BEGIN
        ITEM    FIX    A    24    S    5    $
            BEGIN
                12.5A5    6Ø.3A5    27.4A5
            END
        ITEM    NAME    H    3    $
            BEGIN
                3H(BOS)3H(PSM)3H(JAX)3H(NYC)
                3H(WDC)3H(ALB)3H(CHI)
            END
    END
```

The value of the fixed point item FIX in the first three entries of the unamed table will be preset to the values indicated. The values of the hollerith item NAME in the first seven entries of the table will be preset to the names indicated.

8. TABLE STRUCTURE EXAMPLE

An example of processing variable length tables with programmer designated structure:

```
TABLE    INFO    V    4ØØ    S    2    $

    BEGIN

        ITEM    SHIP    S    V(CAR)    V(TANK)        V(SUB)

                             V(DES)    Ø     Ø    $

        ITEM    SHIPID       H    3    Ø     8    $

        ITEM    AIRCFT       I   16    U     1   16    $

        ITEM    TORPED       I    8    U     1   24    $

        ITEM    CARCAP       I   24    U     1    8    $

        ITEM    MAXRNG       F    1    Ø     $

    END
```



The preceding table INFO is designed to make more efficient use of storage space when the items AIRCFT, TORPED, CARCAP, and MAXRNG apply to the statuses V(CAR), V(SUB), V(TANK), and V(DES) respectively, and to the statuses V(CAR), V(SUB), V(TANK), and V(DES) respectively, and are mutually exclusive. Thus, for a tanker, V(TANK), only the cargo capacity, CARCAP, is of interest. Therefore, the other items do not apply and there is no need to reserve separate storage space for them in each entry of the table.

For example, to obtain the number of aircraft in a fleet composed of carriers, tankers, destroyers, and submarines using the above table:

```
AIR = Ø  $

FOR I = Ø, 1, NENT(INFO)-1  $

    BEGIN

        IF SHIP ($ I $) EQ V(CAR) $

          AIR = AIR + AIRCFT ($ I $) $

    END
```

A similar table structure could be obtained by using an overlay statement instead of the programmer-designated structure. For example:

```
OVERLAY TORPED = CARCAP = MAXRNG = AIRCFT $
```

In this case, space would be reserved for the largest item (MAXRNG) and all the other items would be right justified within that space.

A constant must be able to fit in a single machine word.

1. ARITHMETIC CONSTANTS

Integer	Written as signed or unsigned decimal values with no decimal point, e.g., +10, 6, -20, 12E3. The 12E3 signifies 12 multiplied by $1Ø^3$, or 12ØØØ. 12E-3 would mean 12×10^{-3}.
Fixed Point	Written as signed or unsigned decimal values with a decimal point, followed by the letter A, followed by the number of fractional BITS desired, e.g., +1Ø.AØ, 5.6A1, -3.18A5E3. Care should be used in specifying the number of fractional bits so that the desired accuracy is achieved.
Octal	Written with the letter O, followed by an octal number enclosed in parentheses, e.g., O(3762). 16 octal digits are the maximum permitted.

2. LITERAL CONSTANTS

Hollerith	Written as a decimal number, followed by the letter H, followed by hollerith characters enclosed in parentheses. The number of hollerith characters (including spaces) must be the same as the decimal number preceding the H, e.g., 3H(MON) 7H(JAN-DEC) 8 letters are the maximum permitted.
Standard Transmission Code	Written with the same format as hollerith but using the letter T, e.g., 3T(MON) 8 characters are the maximum permitted.

3. STATUS CONSTANTS

Status	Used to set an item equal to the value of a status, e.g., TAPSTAT = V(READY) where READY must have been declared as a status value for TAPSTAT previously.

1. STATEMENT LABELS

A statement label may be the same as an item name and has the same restrictions described for identifiers in Section 1.1c. Statement labels must not be modified arithmetically either in the labeled statement or in a statement referring to the labeled statement. The statement label must be followed by a period in the statement which it labels. The period is not used in statements referring the the labeled statement, for example:

```
GOTO    CHECK    $
          .
          .
          .
CHECK. XX = YY + ZZ $
```

A statement may have more than one label. Each label must be follwed by its own period.

2. ASSIGNMENT STATEMENTS

The assignment statement consists of a variable followed by equal sign (=) followed by a variable, a constant or a numeric formula. The assignment statement is written in the form of an equation, but it is not an equation. Any arithmetic indicated in the right-hand side of the statement is performed and the resulting value is assigned to the variable on the left-hand side. No arithmetic can be performed in the left-hand side. The left-hand side denotes a single variable value. Mixtures of item types (e.g., floating, integers, arithmetic) are permissible and any necessary conversions implied will be performed by the compiler. If the computed value of the right-hand side is not the same numeric form as the variable on the left, the value will be converted before it is assigned. For example:

```
XX = YY $

ZZ($A$) = AA + BB - CC*DD $
```

Literal assignment is also possible. In executing literal assignment statements there will not be any conversion among hollerith, transmission code, and octal values. The value of the right-hand side must be the same length as the literal variable on the left-hand side. For example:

```
STATE = 5H(READY) $

STATE = 5H(NO GO) $
```

In a status assignment statement, if the right-hand side contains a status constant, it must be one of those previously declared for the variable in the left-hand side. Otherwise there is no way for the compiler to associate a value with the status constant. For example:

```
WEATHER = V(CLOUDY) $

CARD    = V(SPADE)  $
```

3. UNCONDITIONAL TRANSFER STATEMENTS (GOTO)

The unconditional transfer is generated by the JOVIAL primitive GOTO followed by a statement label, for example:

```
GOTO    CHECK    $
```

The next instruction operated by the program will be the statement labeled CHECK.

4. PROGRAM TERMINATION STATEMENTS (STOP)

A machine halt is generated by the statement:

```
STOP$
```

An additional use of this statement is:

```
STOP    XX$
```

where XX is a statement label. In this case, the program will continue at the statement labeled XX, when the computer is restarted.

5. DECISION STATEMENTS

Decision statements are used in decision-making processes where two or more alternatives are available. Decision statements are divided into conditional statements (IF) and alternative statements (IFEITH, ORIF) and involve the use of relational operators and logical operators. A decision statement must not be followed immediately by another decision statement.

RELATIONAL OPERATORS

The JOVIAL relational operators are:

EQ equal to

NQ not equal to

GR greater than

LS less than

GQ greater than or equal to

LQ less than or equal to

LOGICAL OPERATORS

The JOVIAL logical operators are:

AND

OR

NOT

(1) CONDITIONAL STATEMENTS (IF)

Conditional statements are introduced by the JOVIAL primitive IF. If the relational expression following the IF is true, the next statement is executed. If it is false, the next statement is skipped. For example:

```
IF  XX  GR  AA  $
  GOTO  A1  $
XX  =  XX  +1  $
```

If the value of XX is greater than that of AA, the GOTO statement will transfer operation to the statement labeled A1; otherwise XX will be incremented by 1, etc.

An IF statement may include a string of relational and logical operators. For example:

```
IF  ABLE  EQ  BAKER  AND  TEMP  EQ  Ø  $
                    or
IF  ABLE  LS  BAKER  LQ  1Ø  $
```

When using AND, each part of the statement must be true for the whole statement to be true. When using OR, if either expression is true, the whole statement is true.

The JOVIAL system processes IF statements from right to left. If parentheses are used, they are interpreted first within any expression. For example:

```
IF  ABLE  EQ  BAKER  AND  (TEMP  EQ  Ø  OR  PRESS  LS  14.7)  $
```

Arithmetic may also be performed in IF statements, for example:

```
IF  XX-3*  TEMP**4  LS  YY*2  $
```

Expressions used with relational operators should be consistent in scaling. Floating, fixed, integer, literal, etc., values should not be mixed, as opposed to normal arithmetic statements.

(2) ALTERNATIVE STATEMENTS (IFEITH, ORIF, END)

Alternative statements are introduced by the JOVIAL primitives IFEITH or ORIF. The IFEITH introduces a set of alternative statements which consists of the initial IFEITH followed by a non-decision statement, optionally followed by ORIFs alternating with non-decision statements, and terminated by an END. For example:

```
      IFEITH   ALPHA   LS   BETA   $
               ALPHA   =    BETA   $
A1.   ORIF     ALPHA   +    BETA   GR   1Ø   $
      A2.  BEGIN
               GAMMA   =    (ALPHA + BETA)   /   2   $
               ALPHA   =    GAMMA + 1   $
               BETA    =    GAMMA + 1   $
           END
           ORIF   ALPHA   EQ   1   $
              GOTO   ERROR   $
           END
      A3.  ALPHA   =   Ø   $
```

Each alternative is tested in turn; testing may start with the IFEITH or by branch to a labeled (A1) ORIF. The conditions expressed after the IFEITH or ORIF are tested and, if false, control passes to the next alternative. In the example, if the IFEITH is tested first and found false, the ORIF labeled A1 would be tested next. If an alternative statement is tested and found true, the non-decision statement immediately following the true alternative statement is executed; if this execution does not cause control to pass out of the set of alternatives (in the example, see the compound statement labels A2), the remaining alternatives will be skipped when execution of the non-decision statement is complete and control will pass to the statement following the terminating END (in the example, control would pass to the statement labeled A3).

All rules for the evaluation of the conditions following the IFEITH and ORIF in the alternative statements are identical to those applying to IF statements.

6. LOOP STATEMENTS (FOR)

(1) LOOP VARIABLES

A loop variable is a single letter variable which is preset by a FOR statement.

For example: obtain the sum of all XX entries in a 7-entry table:

```
SUM = Ø $
FOR I = Ø, 1, 6 $
    SUM = SUM+XX($ I $) $
```

The right term of the FOR statement defines the indexing of the value of I in the loop. That is, the initial value of I is Ø; after each pass through the loop, I will be incremented by 1, and the last pass through the loop will be made with I equal to 6.

The reverse of the above procedure may also be used, for example:

FOR I = 6, -1, Ø $

A numeric formula may be substituted for any of the three constants to the right of the =.

When the number of entries in a table is not known or may change for subsequent compilation and the entire table is to be processed, the JOVIAL primitive NENT may be used. NENT followed by a table name or the name of a table item will automatically obtain the number of entries in the table or the number of entries of an item contained in the table, for example:

NENT (TAB1)

will give the number of entries in the table or the number of entries in the table containing TAB1, depending on whether TAB1 is declared as a table or an item.

To process all entries in a table by using a loop variable, the number of entries must be reduced by 1 to account for the convention of starting a table with entry Ø, for example:

FOR I = Ø, 1, NENT(TAB1) -1$

Another method of processing a complete table is to use the JOVIAL primitive ALL, for example:

FOR I = ALL(TAB1) $

In this case, the initial value of I will be set to the number of entries minus 1, and will be decreased by 1 for each pass through the loop, down to and including entry Ø. The same effect is achieved with ALL (item name).

A similar use may be made of the JOVIAL primitive NWDSEN, signifying the number of words per entry, for example:

NWDSEN (TAB1) for a table

NWDSEN (XX) for a table item

(2) CONTROL

Loops are controlled by FOR statements primarily. However, the value of a loop variable (and therefore the control of the loop) may be altered by an assignment statement resetting the loop variable within the loop. The value of a loop variable assigned by a FOR statement is only valid for the JOVIAL (compound) statement following the FOR statement. Otherwise, it is undefined. A compound statement bracketed by BEGIN ....END primitives is frequently used following a FOR statement, for example:

```
FOR I = Ø, 1, 499 $
    BEGIN
        XX($ I $) = Ø $
        YY($ I $) = Ø $
    END
```

This loop sets the first 5ØØ values of items XX and YY to zero.

If several FOR statements precede a loop, the first of these controls the loop, for example:

```
FOR A = Ø, 2, 1ØØ $
    FOR B = 12, 3 $
        FOR C = 6Ø, -1, 25 $
            FOR D = 7
                BEGIN
                .
                .
                .
                END
```

This loop will operate until the test value for A equals 1ØØ. However, each time a pass is made through the loop, the values of A, B, and C will be incremented. The value of D will remain at 7 since no increment has been specified. The test value specified for C will be ignored since the first FOR statement controls the loop.

If the first FOR statement does not contain a test value, no automatic test will be made for the loop and some other means, such as a GOTO statement, must be used to terminate the loop.

Similarly, a FOR statement may have a single factor, for example:

```
FOR I = Ø $
```

In this case, incrementing must b accomplished by some other means such as an assignment statement.

(3) NESTING

Nested FOR statements generate loops within loops, for example:

```
SAMPLE = Ø $

FOR I = Ø, 1Ø, 49 $

   BEGIN

      FOR J = I, 1, I+9 $

         BEGIN

            ABLE($ J $) = BAKER($ J $) $

         END

      SAMPLE = SAMPLE + ABLE($ I $) $

   END
```

In this case, the values of ABLE are set to the values of BAKER, in groups of ten. The value of SAMPLE is the sum of every 1Øth value of ABLE, beginning with the first value of ABLE, then the eleventh, 21st, etc. Note that the BEGIN...END brackets of the internal loop are not necessary since there is only one statement in the loop.

In some cases, it may be desirable to exit from an inner to an outer loop, or to increment a loop variable, if some condition occurs before a complete pass has been made. This is accomplished by using the JOVIAL primitive TEST, for example:

```
FOR I = Ø, 1Ø, 49 $

   BEGIN

      FOR J = I, 1, I+9 $

         BEGIN

            IF BAKER($ J $) EQ Ø $

               TEST I $

            ABLE($ J $) = BAKER($ J $) $

         END

      SAMPLE = SAMPLE + I $

   END
```

Whenever the value of BAKER($ J $) is equal to Ø, the TEST I statement will operate and will cause the program to skip to the end of the loop controlled by I, whereupon I will be incremented and, if I has not exceeded 49, another pass made. If a loop variable is not specified after TEST, the program will increment the loop variable of the innermost loop containing the TEST statement.

Note that the value of aloop variable is always incremented BEFORE making the test at the end of the loop.

7. COMPOUND STATEMENTS (BEGIN...END)

A compound statement is a group of JOVIAL statements bracketed by the JOVIAL primitive BEGIN and END. The compound statement is treated as if it were a single JOVIAL statement. A common use for the compound statement is to perform a series of calculations if an IF statement is true, for example:

```
      IF   ABLE   EQ   7   $
         BEGIN   RST   =   6   $
                 XYZ   =   3   $
                 GOTO   A1   $
         END
         RST   =   7   $
         XYZ   =   4   $
A1.   STOP   $
```

In this case, the three statements following BEGIN will be executed if the IF statement is true. Otherwise, the statements following END will not be executed following the IF statement. Note that BEGIN and END are not followed by the $.

Another example of a compound statement:

```
      IF PLANE EQ V(HOSTILE) $
         BEGIN
            ALARM = V(ON) $
            GOTO XX $
         END
      ALARM = V(OFF) $
            .
            .
            .
XX.   STOP $
```

JOVIAL arithmetic is performed using arithmetic operators and variables. Parentheses may be used within a JOVIAL arithmetic expression (also called a numeric formula), for example:

```
AA*(CC+EE) + GG
```

Arithmetic operators of the same level are processed from left to right. When parentheses are used, the expressions within them are processed in order, starting at the innermost parentheses.

The levels of JOVIAL arithmetic, in descending order of processing priority, are:

(1) Negation

(2) Exponentiation

(3) Multiplication

(4) Division

(5) Addition

(6) Subtraction

Negative exponents are legal. Negative roots and division by zero will give unpredicable results. Each arithmetic operation involving a floating point value is performed in floating point.

ARITHMETIC OPERATORS

The following arithmetic operators are used in JOVIAL:

+	add
-	subtract
*	multiply
/	divide
** or (* *)	exponentiation
ABS ( ) or (/ /)	absolute value

Examples of JOVIAL arithmetic:

(1) $\frac{a^2 - 4bc}{c} + 2c \frac{-8ab}{c}$

may be written as

```
(AA**2-4*BB*CC) /CC + 2*CC - 8*AA*BB/CC
```

(2) $ax^4 + bx^3 + cx^2 + dx + e$

may be written as

```
AA*XX**4+BB*XX**3+CC*XX**2*DD*XX+EE
```

or, by factoring,

```
( ( (AA*XX+BB)*XX+CC)*XX+DD)*XX+EE
```

Factoring of an algebraic expression usually reduces the amount of coding produced by a JOVIAL statement.

1. BIT

In order to extract one or more bits from an item, the JOVIAL primitive BIT is used. The starting bit of a group of bits, the number of bits to be extracted, and the item all must be specified. If only a single bit is to be extracted, the number of bits need not be specified.

For example:

```
BIT($  Ø  $)   (NUM($  I  $))
```

This expression will extract the first bit of item NUM($ I $).

For example:

```
BIT($  Ø,  4  $)   (AMOUNT)
```

This expression will extract the first four bits of item AMOUNT.

For example:

```
BIT($   7,   3   $)    (NAME)
```

This expression will extract three bits starting at the eight bit (bit #7) of item NAME.

The starting bit is relative to the high order bit of the item specified, not of the word in which the item is stored.

If more than one bit is extracted, no check is made by the compiler to determine that all the bits are within the item specified or that the end of the word is not exceeded.

An example of extracting bits from one item and depositing them in another item:

```
BIT($  0,   4   $)   (AMOUNT) = BIT($   Ø,   4   $)   (NUM($   I   $))   $
```

2. BYTE

The BYTE modifier is used in exactly the same way as the BIT modifier except that the first number indicates character position and the second indicates number of characters. The BYTE modifier causes 6-bit bytes to be extracted from the item specified. BYTE modifiers are treated as literal constants when used in arithmetic expressions.

For example:

```
BYTE   ($   Ø,   2   $)   (IDENT($   I   $))
```

This expression will extract the first twelve bits of IDENT($ I $).

For example:

```
IF BYTE ($ Ø $)  (NAME($ J $))  EQ 1H(B) $
GOTO  READ  $
GOTO SKIP    $
```

For example, conversion of a hollerith number to a decimal number:

```
DEC = Ø $
FOR J = Ø, 1, 3 $
   DEC = DEC+BYTE($3-J$)  (NUM)*1Ø**J$
```

Note that any arithmetic performed in the BYTE modifier (e.g., ($3-J$) above) must have a positive value. Any fractional values will be truncated to give an integer.

A JOVIAL switch is used in cases where multiple branching is desired. For example, a switch could be used instead of the following sequence:

```
IF ABLE EQ Ø $
   GOTO XX $
IF ABLE EQ 1 $
   GOTO YY $
IF ABLE EQ 2 $
   GOTO ZZ $
```

1. ITEM SWITCH

An item switch is declared by specifying the primitive SWITCH, the switch name, the item to switch on, the item values to be tested, and the statement labels associated with the item values. For example, a switch to accomplish the above multiple branching sequence would be written:

```
SWITCH BACK (ABLE) = (Ø=XX, 1=YY, 2=ZZ) $
```

where BACK is the switch name.

Whenever it is desired to branch to one of the specified statement label locations based on the value of able, the statement GOTO BACK $ is used. If none of the conditions of the switch are met, the program will continue at the statement following the GOTO statement. Caution should be used in locating a switch declaration within a JOVIAL program. Since the declaration causes the generation of code, it should be located so that control cannot "fall" into it during normal execution of the program.

A switch name may be subscripted if the item to switch on is a table item. The switch declaration format is the same as that referencing a simple item (see last example). For example:

```
FOR I = Ø, 1, 1Ø, $
   GOTO BACK ($ I $) $
```

In this case, the test will be made on ABLE ($ I $).

2. INDEX SWITCH

An index switch is declared using the primitive SWITCH, the switch name, and a list of statement labels, for example:

```
SWITCH BACK = (AA, BB, CC, DD, XX) $
```

An index value is assigned to each statement label, beginning with Ø and increasing by 1 for each label specified. Thus, in the example above, the index value of AA is Ø, BB is 1, XX is 4. This switch is referenced using a GOTO statement containing the index switch name, for example:

```
GOTO BACK ($ I $) $
```

If the value of I is 2, then, in the above example, the program would branch to the statement labeled CC.

The value of the index must not exceed the index values contained in the switch declaration: if it does, the results are unpredictable.

Index values without associated statement labels may be included in a switch declaration by omitting the statement label but including the comma. Such index values will cause the program to return to the statement following the statements which referenced for the switch, for example:

```
SWITCH BACK = (AA,,CC,,DD) $
```

If this switch is entered by the statement

```
GOTO BACK ($ I $) $
```

then, for the cases when I has a value of 1 or 3, the program will return to the statement following the GOTO statement.

1. LOCATION ('LOC)

The JOVIAL location function, 'LOC gives the address of a table, item, statement label or program when declared using the format:

```
'LOC  (NAME)
```

Note that the address is the absolute core address.

Program names and statement labels must be followed by a period. The 'LOC function for a table will give the address of the first word of the table (excluding any control words). The 'LOC function for a subscripted variable will give the address of the first occurrence of the variable in the table.

The 'LOC function may be used in any numeric formula.

2. DEFINE PRIMITIVE

The JOVIAL primitive, DEFINE, is used to set the value of an item. Once defined, an item may be redefined by a new DEFINE statement but may not be "undefined." A JOVIAL primitive may not be redefined. A DEFINE statement equates a name with a numerical value bracketed by double primes ("). The name may then be used for the value in subsequent statements.

For example:

```
DEFINE   PI   "3.14159"   $
```

which might then be used as

```
C=2*PI*RAD$
```

1. FUNCTIONS (PROC)

A function is used to obtain a single output value from a compound JOVIAL statement. Functions simplify JOVIAL programs by permitting frequently used subroutines to be specified once and to be called using the function name.

A function is declared using the JOVIAL primitive PROC, followed by the function name and a dummy parameter. Following the PROC statement is a list of items and a compound statement which is the body of the function. The item list must include the function name. Functions may be declared externally to the computer program in compools and libraries. There is no difference in their use when externally declared from their use when internally declared. The Control Card provides the means to make external declarations available to computer programs.

For example, a square root function could be declared in the following way:

```
PROC   SQRT(SQUAR)   $
   ITEM   SQRT   F   $
   ITEM   SQUAR   F   $
BEGIN
   SQRT = .5*SQUAR   $
A2. IF(/SQRT**2 - SQUAR/) GR .ØØØ1 $
   BEGIN
      SQRT = .5*(SQRT + SQUAR/SQRT) $
      GOTO A2 $
   END
END
```

A function is called by the function name followed by the input parameters. The function call must be within a JOVIAL statement. The above SQRT function would be called by the following statement:

```
HYPOT = SQRT(LSIDE**2) $
```

The function would be entered with the value in the parentheses used as the value for the dummy parameter SQUAR in the function. When execution of the function has been completed, the program returns to the statement which called the function. In the example cited, the value obtained from the SQRT function would then be entered into the location of HYPOT.

2. PROCEDURES (PROC)

A procedure is the general case of JOVIAL closed subroutines. A procedure is defined by the primitive PROC, followed by input parameters, output parameters, an item list, and a compound statement. The item list must not contain the procedure name. Procedures may be declared externally to the computer program in compools and libraries. There is no difference in their use when externally declared from their use when internally declared. The Control Card provides the means to make external declarations available to computer programs.

A procedure can have multiple input and output parameters, or it may have input only, output only, or no parameters at all. The last case may be represented by empty parentheses following the procedure name or by the absence of parentheses. For example:

```
PROC    SQRT(SQUARE = SQT)    $

   ITEM    SQT    F    $

   ITEM SQUARE  F    $

   BEGIN
      .
      .
      .
   END
```

A procedure is called by a statement beginning with the procedure name and followed by the input and output parameters, if any. For example:

$$c = a^2 + b^2$$

might be represented by

```
SQRT(LSIDE($ I $)**2 + RSIDE ($ Ø $)**2 = HYPOT ($ I $)) $
```

The return from a procedure is to the statement following that which called the procedure.

3. CLOSE ROUTINE

```
CLOSE
```

This type of JOVIAL subroutine, declared by a CLOSE statement, is a parameterless closed subroutine, for example:

```
CLOSE BLANK $

   BEGIN

      FOR Z = 14, -1, Ø $

         WRITE($ Z $) = 8H(      )$

   END
```

The above CLOSE routine would be used to set a table to all blanks, as follows:

```
TABLE   IMAGE   R   15   $
  BEGIN
    ITEM   WRITE   H   8   $
  END
    .
    .
    .
  GOTO BLANK   $
```

Control should be passed to the closed subroutine only by a GOTO statement; if control reaches a closed subroutine by other means, the results are unpredictable. Return from the CLOSE subroutine is to the statement following the calling GOTO.

The CLOSE routine must be compiled as a part of the symbolic deck of the program by which it is called.

4. 'PROGRAM

Another type of JOVIAL closed subroutine is that referenced by the 'PROGRAM declaration. This type of subroutine is also called by a GOTO statement and return is to the statement following GOTO. The declaration has the following format:

```
'PROGRAM   Name   Optional Core Location   $
```

The 'PROGRAM subroutine must not be compiled as a part of the symbolic deck of the program by which it is called; if a core location does not appear in the declaration, a core location must be assigned to the 'PROGRAM name in the compool.

5. TERMINATION (RETURN)

The RETURN operator may be used in any function, procedure or closed subroutine to terminate operation and return control to the calling program from any point within the function, procedure or closed subroutine.

Direct coding permits the programmer to include AN/GSA-51 or AN/GSA-51A machine language coding within a JOVIAL language program (see CGTM2387A, Volume II, Chapter I).

1. BRACKETS (DIRECT, JOVIAL)

Machine language coding must be preceded by the JOVIAL primitive DIRECT and is terminated with the JOVIAL primitive JOVIAL, each of which must be contained alone on separate cards.

2. ASSIGNMENT (ASSIGN)

Within a direct coding section of a program, tables, items, NENT or 'LOC may be referred to only by the use of an ASSIGN statement.

```
DIRECT   $
   :
   :
ASSIGN   A(4) = XX   $
   :
   :
ASSIGN   YY(S II S) = H (   )   $
   :
   :
JOVIAL   $
```

The first ASSIGN statement causes the value of XX to be loaded into the stack. The A preceding the parenthses indicates normal stack. An H would indicate hold stack and an N would also indicate normal stack. However, the normal or hold position of the stack is not guaranteed. The number within the parentheses indicates the number of fractional bits if XX is a fixed point constant. If a zero appears within the parentheses, then XX must be an integer or a non-numberic item. A blank within the parentheses indicates that XX is a floating point number. The second ASSIGN statement causes the contents of the stack to be loaded into item YY, without moving the stack.

3. CODE

All of the codes below are described in SDC TM-1350/004 except LBL; all are also described in MITRE SR-91 except LBL, SSS (which is identical in function to SSD), and SSF (which is identical in function to SSU).

LBL is used to assign a symbolic tag to the code associated with a JOVIAL card or an ASSIGN card. For example:

```
Cols    8-12    14-18    19-ff

    SETA    LBL

            ASSIGN AA = H(+4) $
                :
                :
            UCT    SETA
```

will allow another portion of the program to branch to the ASSIGN statement which sets item AA.

The following machine codes are legal:

ACE	BSU	FDV	LOR	SRR
ACG	CBF	FMU	LSR	STF
ACL	CEF	FSU	LTF	SSD
AIF	CEQ	HLT	LXF	SSF
BAD	CGF	IRR	LXR	SSS
BAF	CGR	LAF	NOP	SSU
BDV	CLA	LAN	RPT	TIO
BMU	CLF	LCF	RVS	TRM
BRB	CLS	LCM	SAF	TRS
BRC	CSE	LOF	SER	UCT
BSF	FAD		SRJ	XLC

The following declarative code is legal:

OCT

The following pseudo codes are legal:

ALC	ARS	FLSD
ALCD	ARSD	FRC
ALS	CYC	FRCD
ALSD	FLC	FRS
ARC	FLCD	FRSD
ARCD	FLS	

The following control codes are legal:

blank

LBL

SET (BPR, only)

4. CARD FORMAT

Columns 8-12 are reserved for location tags. If these columns are blank, the operation code of the instruction may be assigned to any one of four positions (bits 1-12, 13-24, 25-36, 37-48) in the 48-bit word generated by the compiler. If these columns are not blank, the operation code will lie in bits 1-12; if these columns contain an *, symbolic tag is associated with the instruction, and if they contain a location tag, the instruction may be symbolically referenced elsewhere in the direct code area by the location tag.

Columns 14-18 are reserved for machine codes, declarative codes, pseudo codes, and control codes.

Columns 19 and ff are reserved for operands. All operands are coded as described in TM-1350/004 with the following exceptions for the memory syllable:

(1) R-C words do not include D( ). A Decimal integer is coded A (±#). A fixed point number with a fractional part is coded with a sign, a set of numbers, a decimal point and the letter "A" followed by an integer specifying the desired number of binary bits to the right of the binary point. The fixed point number is enclosed in parentheses and preceded by an "A". A(+5.7A8)

(2) A Hollerith R-C word is coded #H( ) with # equal to number of characters inside the parentheses.

(3) No indirect addressing is allowed on system tables (index applied because of compool definition).

(4) No increments or decrements are allowed in indirectly addressed tables or items.

The statements and declarations described in the preceding paragraphs provide for the construction of programs, except for the definition of the beginning and end. This definition is provided by the JOVIAL primitives START and TERM S. Except for the control card a card beginning with

```
START
```

must be the first card of a JOVIAL program deck, and a card ending with

```
TERM optional statement label $
```

must be the last card of the deck. If the optional statement label is omitted, execution of the program will begin with the first executable statement which is not part of a procedure/function/close[1] routine declaration. The presence of the optional statement label causes execution of the program to begin with the statement bearing the indicated label. For example:

```
START
    ITEM COUNT  I  1Ø  U  $
    COUNT = Ø  $
    FOR I = Ø, 1, 99  $
        COUNT = COUNT + 1  $
    STOP  $
TERM  $
```

and its equivalent

```
START
    ITEM COUNT   I   1Ø   U   $
AA. STOP     $
BB. COUNT = Ø   $
    FOR I = Ø, 1, 99  $
        COUNT = COUNT + 1   $
    GOTO  AA  $
TERM  BB  $
```

Symbolic decks may or may not be headed by a Control Card. The compiler will accept decks which have been prestored on tape, in which case a Control Card is required with the keyword INPUT identifying the tape drive, as well as accepting decks input directly through the card reader.

---

[1] Note that only close routines declared by CLOSE must be compiled in the calling program, and that those declared by 'PROGRAM must be compiled separately.

NOTES

NOTES

SAVE A LIFE

If you observe an accident involving electrical shock,
DON'T JUST STAND THERE - DO SOMETHING!

RESCUE OF SHOCK VICTIM

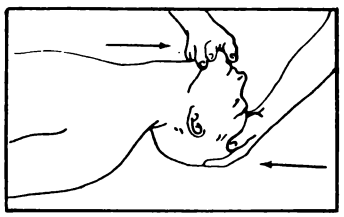
The victim of electrical shock is dependent upon you to give him prompt first aid. Observe these precautions:

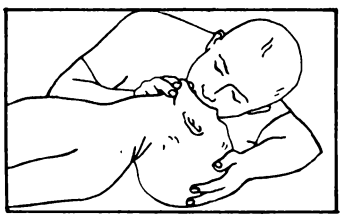
1. Shut off the high voltage.
2. If the high voltage cannot be turned off without delay, free the victim from the live conductor. REMEMBER:
   a. Protect yourself with dry insulating material.
   b. Use a dry board, your belt, dry clothing, or other non-conducting material to free the victim. When possible PUSH - DO NOT PULL the victim free of the high voltage source.
   c. DO NOT touch the victim with your bare hands until the high voltage circuit is broken.

FIRST AID

The two most likely results of electrical shock are: bodily injury from falling, and cessation of breathing. While doctors and pulmotors are being sent for, DO THESE THINGS:

1. Control bleeding by use of pressure or a tourniquet.
2. Begin IMMEDIATELY to use artificial respiration if the victim is not breathing or is breathing poorly:
   a. Turn the victim on his back.
   b. Clean the mouth, nose, and throat. (If they appear clean, start artificial respiration immediately. If foreign matter is present, wipe it away quickly with a cloth or your fingers).





   c. Place the victim's head in the "sword-swallowing" position. (Place the head as far back as possible so that the front of the neck is stretched).
   d. Hold the lower jaw up. (Insert your thumb between the victim's teeth at the midline - pull the lower jaw forcefully outward so that the lower teeth are further forward than the upper teeth. Hold the jaw in this position as long as the victim is unconscious).
   e. Close the victim's nose. (Compress the nose between your thumb and forefinger).
   f. Blow air into the victim's lungs. (Take a deep breath and cover the victim's open mouth with your open mouth, making the contact air-tight. Blow until the chest rises. If the chest does not rise when you blow, improve the position of the victim's air passageway, and blow more forcefully. Blow <u>forcefully</u> into <u>adults</u>, and <u>gently</u> into <u>children</u>.
   g. Let air out of the victim's lungs. (After the chest rises, quickly separate lip contact with the victim allowing him to exhale).
   h. Repeat steps f. and g. at the rate of 12 to 20 times per minute. Continue rhythmically without interruption until the victim starts breathing or is pronounced dead. (A smooth rhythm is desirable, but split-second timing is not essential).

DON'T JUST STAND THERE - DO SOMETHING!