File No. 1410/7010-24
GC28-0327-5

IBM Systems Reference Library

IBM 1410/7010 Operating System (1410-PR-155)

COBOL—1410-CB-969

This publication is designed to be used by programmers in conjunction with the publication, *IBM General Information Manual, COBOL,* Form F28-8053, and contains additional specifications required to write COBOL programs to be processed under the 1410/7010 Operating System.

MAJOR REVISION (July 1966)
This publication is a major revision of *IBM 1410/7010 Operating System; COBOL*, Form C28-0327-4, which is now obsolete. Changes are indicated by a vertical bar at the left of the corrected text, or by a bullet (•) at the left of the figure caption.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Address comments concerning the contents of this publication to:
IBM Corporation, Programming Systems Publications, Department 637, Neighborhood Road, Kingston, New York 12401

Contents

Acknowledgment

In accordance with the requirements of the official government manual, *COBOL-1961-Extended,* Form number 1962-0668996, describing COBOL (obtained by sending a purchase order and $1.25 to: Superintendent of Documents, U. S. Government Printing Office, Washington 25, D. C.), the following extract from that manual is presented for the information and guidance of the user:

"This publication is based on the COBOL System developed in 1959 by a committee composed of government users and computer manufacturers. The organizations participating in the original development were:

Air Materiel Command, United States Air Force
Bureau of Standards, United States Department of Commerce
Burroughs Corporation
David Taylor Model Basin, Bureau of Ships, United States Navy
Electronic Data Processing Division, Minneapolis-Honeywell Regulator Company
International Business Machines Corporation
Radio Corporation of America
Sylvania Electric Products, Inc.
UNIVAC Division of Sperry Rand Corporation

"In addition to the organizations listed above, the following other organizations participated in the work of the Maintenance Group:

Allstate Insurance Company
The Bendix Corporation, Computer Division
Control Data Corporation
E. I. du Pont de Nemours and Company
General Electric Company
General Motors Corporation
Lockheed Aircraft Corporation
The National Cash Register Company
Philco Corporation
Royal McBee Corporation
Standard Oil Company (New Jersey)
United States Steel Corporation

*Trademark of Sperry Rand Corporation

"This COBOL-61 manual is the result of contributions made by all of the above-mentioned organizations. No warranty, expressed or implied, is made by any contributor or by the committee as to the accuracy and functioning of the programming system and language. Moreover, no responsibility is assumed by any contributor, or by the committee, in connection therewith.

"It is reasonable to assume that a number of improvements and additions will be made to COBOL. Every effort will be made to insure that the improvements and corrections will be made in an orderly fashion, with due recognition of existing users' investments in programming. However, this protection can be positively assured only by individual implementors.

"Procedures have been established for the maintenance of COBOL. Inquiries concerning the procedures and the methods for proposing changes should be directed to the Executive Committee of the Conference on Data Systems Languages.

"The authors and copyright holders of the copyrighted material used herein: FLOW-MATIC*. Programming for the UNIVAC* I and II, Data Automation Systems © 1958, 1959, Sperry Rand Corporation; IBM Commercial Translator, Form No. F28-8013 copyrighted 1959 by IBM; FACT, DSI 27A5260-2760, copyrighted 1960 by Minneapolis-Honeywell, have specifically authorized the use of this material, in whole or in part, in the COBOL specifications. Such authorization extends to the reproduction and use of COBOL specifications in programming manuals or similar publications.

"Any organization interested in reproducing the COBOL report and initial specifications, in whole or in part, using ideas taken from this report or utilizing this report as the basis for an instruction manual or any other purpose is free to do so. However, all such organizations are requested to reproduce this section as part of the introduction to the document. Those using a short passage, as in a book review, are requested to mention 'COBOL' in acknowledgment of the source, but need not quote this entire section."

Purpose of this Publication

This publication is designed to be used by programmers in conjunction with the publication, *IBM General Information Manual, COBOL,* Form F28-8053. This publication contains additional specifications required to write COBOL programs to be processed under the 1410/7010 Operating System.

COBOL Source Programs

The similarity between COBOL and ordinary business English facilitates writing COBOL source programs. Source program statements are translated directly into machine language by the COBOL compiler, which takes full advantage of the capabilities of the IBM 1410 and 7010 Data Processing Systems.

Prerequisite and Related Information

A basic knowledge of both COBOL and the IBM 1410/7010 Operating System is required to fully understand the information presented in this publication.

Anyone without this prior knowledge is requested to read the following publications:

IBM General Information Manual, COBOL, Form F28-8053

IBM 1410/7010 Operating System; Basic Concepts, Form C28-0318

IBM 1410 Principles of Operation, Form A22-0526 or *IBM 7010 Principles of Operation,* Form A22-6726

The reader should also be familiar with the contents of the publication *IBM 1410/7010 Operating System; System Monitor,* Form C28-0319. The following 1410/7010 Operating System publications, along with *System Monitor,* should be available for reference purposes: *System Generation,* Form C28-0352; *Basic Input/Output Control System,* Form C28-0322; and *Operator's Guide,* Form C28-0351.

Machine Requirements

The minimum machine requirements for compiling programs using the COBOL compiler are included in the publication, *System Generation,* Form C28-0352. However, machine requirements for running a particular object program depend upon the requirements of the program (for example, the amount of core storage needed).

COBOL Language Forms and Notations

This publication contains all the basic forms of the various verbs, clauses, entries, and other essential elements of the 1410/7010 COBOL language. They are intended to guide the programmer in writing his own statements. If statements are written in formats other than those presented in this manual, the compilation will result in error.

The following rules of notation have been followed in the presentation of these forms:

1. All words printed entirely in capital letters are COBOL words; i.e., words that have preassigned meanings in the COBOL language.

2. All underlined words are required unless the portion of the format containing them is itself optional; i.e., enclosed in square brackets. These are key words and if any such word is missing or is incorrectly spelled, it is an error in the program.

3. All COBOL words not underlined may be included or omitted at the option of the programmer. These words are used only for the sake of readability. Misspelling, however, constitutes an error.

4. All italicized words represent information that must be supplied by the programmer. The nature of the information required is indicated in each case. In most instances, the programmer will be required to provide an appropriate data-name, procedure-name, literal, etc.

5. Material enclosed in square brackets [ ] may be used or omitted as required by the programmer.

6. When material is enclosed in braces { }, only one of the enclosed items is required; the others are to be omitted. The choice is to be made by the programmer.

7. Punctuation, where shown, is essential. Other punctuation may be inserted by the programmer in accordance with the rules specified in the General Information Manual.

8. In certain cases, a succession of operands or other elements may be used in the same statement. This possibility is indicated by the use of three dots following the item affected. The dots apply to the last complete element preceding them; thus, if a group of operands and key words are enclosed within brackets, and three dots precede the closing bracket, the entire group must be repeated if any repetition is required, not merely the last operand.

Identification Division

The information specified in the Identification Division of the source program allows the programmer to identify or label his program, and provide other pertinent information concerning the program. This division must precede the other divisions when the source program is presented to the compiler. The over-all structure of the Identification Division is:

IDENTIFICATION DIVISION.
PROGRAM-ID. *program-name.*

[AUTHOR. *author-name.*]

[INSTALLATION. *any sentence or group of sentences.*]

[DATE-WRITTEN. *any sentence or group of sentences.*]

[DATE-COMPILED. *any sentence or group of sentences.*]

[SECURITY. *any sentence or group of sentences.*]

[REMARKS. *any sentence or group of sentences.*]

Usage of the IDENT* portion of the PROGRAM-ID source statement is explained in the section, "1410/7010 COBOL Compiler Requirements."

For additional details concerning the Identification Division, see the General Information Manual.

NOTE: Special characters, such as the hyphen (-), preceded and followed by blanks, may not be used in the Identification Division.

*Columns 73-80 of the *COBOL Program Sheet* (Reference Format)

In the Environment Division of the COBOL source program the programmer describes to the compiler the physical characteristics of the IBM 1410 or 7010 System that will be used to compile the source program, and the system that will be used to execute the object program. This division must immediately follow the Identification Division when the source program is submitted to the compiler.

Structure of the Environment Division

The over-all structure of the Environment Division for a source program is given below:

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.
OBJECT-COMPUTER.
SPECIAL-NAMES.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
I-O-CONTROL.
```

Each of the subdivisions of the Environment Division is discussed in the following pages. This discussion is in terms of the IBM 1410/7010 COBOL compiler, and therefore includes specifications not contained in the General Information Manual.

Configuration Section

The three paragraphs of the Configuration Section specify, respectively, the computer on which the COBOL compiler is to be run, the computer on which the object program is to be run, and the names of the machine devices and switch conditions referred to by the programmer in the Procedure Division of his source program.

SOURCE-COMPUTER Paragraph

The purpose of this paragraph is to specify the computer on which the COBOL compiler is to run to compile the source program. The general form of this paragaph for the 1410/7010 COBOL compiler is:

SOURCE-COMPUTER. $\left\{\begin{matrix}\underline{\text{IBM-1410}}\\ \underline{\text{IBM-7010}}\end{matrix}\right\}$.

Additional information regarding the source computer (e.g., actual core-storage size, core storage available, etc.) is contained in the Resident Monitor's Communication Region; therefore, no additional entries are needed or permitted in this paragraph. (See *System Monitor.*)

OBJECT-COMPUTER Paragraph

The purpose of this paragraph is to specify the computer on which the object program is to be executed. The general form of this paragraph is:

OBJECT-COMPUTER. $\left\{\begin{matrix}\text{IBM-1410}\\ \text{IBM-7010}\end{matrix}\right\}$.

SPECIAL-NAMES Paragraph

This optional paragraph equates mnemonic-names with device-names representing certain system units or the console printer, and equates condition-names with the status of the system's Standard Input Unit end-of-file switch and/or a switch in the Resident Monitor's Communication Region. The general form of this paragraph is:

SPECIAL-NAMES. [*device-name-1* IS *mnemonic-name-1*
[*device-name-2* IS *mnemonic-name-2* ...]]
[MONITOR-SWITCH
[*literal-1* STATUS IS *condition-name-1*]
[*literal-2* STATUS IS *condition-name-2* ...]]
[I-O-SWITCH EOF-SIU
[ON STATUS IS *condition-name-3*]
[OFF STATUS IS *condition-name-4*]].

DEVICE-NAMES

The device-names of the SPECIAL-NAMES paragraph must be chosen from the following list:

DEVICE-NAME	DESCRIPTION
CONSOLE-PRINTER	the console printer
SYSTEM-OUTPUT-PUNCH	the Standard Punch Unit
SYSTEM-OUTPUT-PRINTER	the Standard Print Unit

System units are discussed in the publication, *System Monitor.*

MONITOR-SWITCH

The MONITOR-SWITCH is used to represent a single-character switch position within the Resident Monitor's Communication Region; the switch is set by the Operator via a $3x console inquiry. The user can in the Procedure Division refer to this switch by means of a condition-name associated with the status of the switch. Literal-1 can be any valid, single-character, non-numeric literal.

After entering the $3x console inquiry, the operator must also enter $50 to cause the system to exit from the Monitor-Switch Wait-Loop routine.

Figure 1 illustrates the use of the MONITOR-SWITCH. SET MONITOR-SWITCH $3X TO A, B, OR C FOR TYPE OF INPUT is the message that will be issued on the console. The operator responds by entering $3A, $3B, or $3C followed by $50. Complete details on the $3x message, and procedures for entering console inquiries, can be found in the *System Monitor* and *Operator's Guide* publications.

I-O-SWITCH EOF-SIU

The I-O-SWITCH EOF-SIU is a programmed switch that indicates the end-of-file status of the Standard Input Unit. This switch can be referred to in the Procedure Division by means of a condition-name associated with the ON or OFF status of this switch.

Figure 2 illustrates a sample SPECIAL-NAMES paragraph.

Input-Output Section

The Input-Output Section of the Environment Division consists of the FILE-CONTROL paragraph and the I-O-CONTROL paragraph.

FILE-CONTROL Paragraph

This paragraph is used to name each file of the source program, identify its medium (i.e., magnetic tape or unit-record equipment) and assign each file to a symbolic unit. Methods of assigning files to magnetic tape and unit-record devices are discussed in that order.

TAPE FILES

The form of the FILE-CONTROL paragraph for files assigned to tape is:

FILE-CONTROL. SELECT *file-name-1*

[RENAMING *file-name-2*]

ASSIGN TO *device-name*

[RESERVE { *integer-1* / NO } ALTERNATE AREA[S]].

[SELECT . . .].

SELECT Clause: Each file to be processed by the object program must be named in a SELECT clause. Each file-name must be unique within the source program, and each file must be described by a File Description entry in the Data Division.

RENAMING Option: The RENAMING option allows the programmer to use the File Description of *file-name-2* in the Data Division for *file-name-1*. This option enables two files to share the same File Description; it does *not* allow the two names to be used interchangeably in the program.

NOTE: The files should be selected in the FILE-CONTROL paragraph in the same relative order as they are described in the FILE SECTION. Select file-name-2 must immediately precede select file-name-1, in the

```
SERIAL  CONT.  A       B
4  6    7     8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
       SPECIAL-NAMES.
           MONITOR-SWITCH 'A' STATUS IS MONITOR-SWITCH-A
                  'B' STATUS IS MONITOR-SWITCH-B
                  'C' STATUS IS MONITOR-SWITCH-C.

       PROCEDURE DIVISION.
       SET-MONITOR-SWITCH-PARAGRAPH.
           STOP 'SET MONITOR-SWITCH #3X TO A, B, OR C
                FOR TYPE OF INPUT'.
           IF MONITOR-SWITCH-A GO TO PROCESS-A.
           IF MONITOR-SWITCH-B GO TO PROCESS-B.
           IF MONITOR-SWITCH-C GO TO PROCESS-C.
       PROCESS-A.
           .
           .
           .
       PROCESS-B.
           .
           .
           .
       PROCESS-C.
```

Figure 1. MONITOR-SWITCH Sample

```
SEQUENCE (PAGE) 1 3 (SERIAL) 4 6  CONT. 7   A 8   B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
                                  SPECIAL-NAMES.
                                      SYSTEM-OUTPUT-PUNCH IS PUNCH
                                      MONITOR-SWITCH '1' STATUS IS MONTHLY-RUN
                                                 '2' STATUS IS WEEKLY-RUN
                                      I-O-SWITCH EOF-SIU ON STATUS IS LAST-CARD.
```

Figure 2. Special-Names Paragraph

FILE-CONTROL paragraph. If both input and output files are involved, the output file must be selected last, and the input file must have an associated File Description. Only one output file may be associated with a given RENAMING clause. The number of alternate areas reserved for each file must be identical.

ASSIGN Clause: Each file must be assigned to a symbolic unit. The device-name in the ASSIGN clause must have the following form for files assigned to tape units:

TAPE-UNIT xxx

TAPE-UNIT is the device name itself and represents a symbolic assignment to magnetic tape. "xxx" is the name of a symbolic unit (e.g., MR1).*

RESERVE Option: This option allows the programmer to specify alternate input or output areas for the implementation of overlap processing. One to five alternate areas per file may be specified (integer-1). If NO ALTERNATE AREA is specified or if the RESERVE option is omitted, overlap processing will not take place.

UNIT-RECORD FILES

The form of the FILE-CONTROL paragraph for files assigned to unit-record equipment is:

FILE-CONTROL. SELECT *file-name-1*

[RENAMING *file-name-2*]

ASSIGN TO *device-name*

[RESERVE { *integer-1* / NO } ALTERNATE AREA[S]].

[SELECT ...].

The SELECT clause, the RENAMING option, and the RESERVE option are used as described for tape files. The device-name of the ASSIGN clause must be chosen from the following list:

DEVICE-NAME	DESCRIPTION
CARD-READER xxx	is the standard device-name for the card reader of the 1402 Card Read Punch, or the 1442 Card Reader. "xxx" is the name of a symbolic unit (e.g., MR1).*
CARD-PUNCH xxx	is the standard device-name for the card punch of the 1402 Card Read Punch. "xxx" is the name of a symbolic unit (e.g., MR2).*
PRINTER xxx	is the standard device-name for the 1403 Printer with 132 print positions. "xxx" is the name of a symbolic unit (e.g., MR3).*

NOTE: The above device-names cannot be used for units assigned as the Standard Input Unit, Standard Punch Unit, and Standard Print Unit for the Operating System.

Figure 3 illustrates a sample FILE-CONTROL paragraph.

```
SEQUENCE (PAGE) 1 3 (SERIAL) 4 6  CONT. 7   A 8   B 12 16 20 24 28 32 36
                                  FILE-CONTROL
                                      SELECT OLD-MASTER-FILE
                                      ASSIGN TO TAPE-UNIT MR2
                                      RESERVE 1 ALTERNATE AREA.
                                      SELECT NEW-MASTER-FILE
                                      RENAMING OLD-MASTER-FILE
                                      ASSIGN TO TAPE-UNIT MR1
                                      RESERVE 1 ALTERNATE AREA.
                                      SELECT TRANSACTION-FILE
                                      ASSIGN TO CARD-READER MRA.
                                      SELECT PRINTER-FILE
                                      ASSIGN TO PRINTER MRB.
```

Figure 3. File-Control Paragraph

I-O-CONTROL Paragraph

The optional I-O-CONTROL paragraph allows the programmer to specify padding of short-length blocks of blocked, fixed-length output records; to control rewinding of tape files; and to establish rerun points.

*The COBOL programmer may reference symbolic units as either xxx or /xxx/. For details concerning symbolic units see the publication, *System Monitor*.

The form of this paragraph is:

```
I-O-CONTROL.[ APPLY literal-1 PADDING ON file-name ]
                                                   [ APPLY... ]
[ APPLY OPEN-WITHOUT-REWIND ON file-name-2 ]
                                                   [ APPLY... ]
[ RERUN EVERY BEGINNING OF REEL OF
                                  { ALL FILES                  } ].
                                  { file-name-1 [file-name-2...] }
```

APPLY Option 1:

APPLY *literal-1* PADDING ON *file-name*

This option is used to specify padding of short-length blocks of a fixed-length, blocked tape-output file. Literal-1 can be any valid, single-character, alphanumeric literal except ⫩, ⫪, °, ƀ, *, √‾, ⫫, and ⁀. If this APPLY option is not specified, the compiler provides padding with spaces where required.

Spaces or nines should be used for padding characters if the file is to be sorted using the 1410/7010 Generalized Tape Sorting Program.

NOTE: The degree sign (°) appears only on special printing chains.

APPLY Option 2:

APPLY OPEN-WITHOUT-REWIND ON *file-name*

This option of the APPLY clause can be used to facilitate the processing of multi-file tape reels. This option only applies to the first reel in which the file is contained; subsequent reels will be rewound.

NOTE: Both APPLY options can be used for a given file.

RERUN Option: This option allows the programmer to specify rerun points (checkpoints) at every beginning of reel of all files, or of selected files. The tape upon which the rerun records are recorded is the optional Core Image file.

If the RERUN clause or Option 2 of the USE verb is used: (1) an MDM file and IOKDF label option D for both 80- and 120-character labels must be included at System Generation, and (2) the file associated with the RERUN clause or Option 2 of the USE verb must be a labeled file. Failure to include these items at System Generation may cause the system symbols /LRA/ and /LRG/ to remain undefined when the COBOL object program is loaded by the Linkage Loader. (System symbols /LRA/ and /LRG/ are explained under the label routines description in the publication *IBM 1410/7010 Operating System; Resident and Transitional Monitors, Input/Output Control System—Programming Systems Analysis Guide*, Form C28-0396.)

If neither the RERUN clause nor Option 2 of the USE verb is used, the undefined system symbols /LRA/ and /LRG/ will not affect the execution of the object program.

Information concerning checkpoints is contained in the publication, *IBM 1410/7010 Operating System; Basic Input/Output Control System*, Form C28-0322. Information concerning restarting a program from a checkpoint is contained in the publication, *IBM 1410/7010 Operating System; Operator's Guide*, Form C28-0351.

The Data Division of a COBOL source program defines the nature and characteristics of the data to be processed by the object program. It begins with the header DATA DIVISION. Each of the three sections of the Data Division also begins with a header, and is followed by the word SECTION as shown below:

```
DATA DIVISION.
FILE SECTION.
  File Description Entries
  Record Description Entries
WORKING-STORAGE SECTION.
  Record Description Entries
CONSTANT SECTION.
  Record Description Entries
```

The File Section describes the input/output files with respect to content and organization. It has two types of entries: the File Description entry, which specifies the physical characteristics and organization of a file; and the Record Description entry, which describes the individual items contained in the data records of the file.

The Working-Storage Section describes the areas of core storage where intermediate results and other items are stored temporarily at object-program execution time.

The Constant Section describes fixed items of data which remain unchanged during the running of the object program.

Any section not required in the program being written should be omitted.

IBM 1410/7010 Files and Records

The programmer should understand how files and records are handled by the IBM 1410/7010 Operating System in order to use the COBOL language effectively in writing the Data Division entries for his source program. Information concerning files and records is therefore given below, prior to discussion of the COBOL language specifications for the Data Division.

Recording Modes

Information in a data processing system may be recorded in various forms and modes. The following discussion pertains to the file-recording modes of the IBM 1410 and 7010 Data Processing Systems. For additional details, see the publication, *IBM 1410 Principles of Operation*, Form A22-0526 or *IBM 7010 Principles of Operation*, Form A22-6726.

EVEN AND ODD PARITY MODES

The IBM 1410 and 7010 can record information on magnetic tape and read information from magnetic tape in either even-parity mode or odd-parity mode.

LOAD AND MOVE MODES

Another 1410/7010 file recording mode specifies how word marks and word separator characters are recorded during read and write operations.

Load Mode: The handling of word marks and word separator characters in the Load mode depends on the type of operation, as follows:

During *write* operations, each word mark is translated into a word separator character that immediately precedes the character with which the word mark was associated in core storage. Each word separator character in storage is translated into two word separator characters on tape.

During *read* operations, word marks already in the input area are cleared. Each word separator character on tape is translated into a word mark associated with the character it immediately preceded on tape, and pairs of word separator characters on tape are translated into single word separator characters without word marks in core storage.

Move Mode: When information is written in the Move mode, word marks have no effect on the data that is recorded on output media. Word marks in storage are undisturbed when information is read in this mode. Each word separator character is read into core storage and written out of core storage as a word separator character.

Standard Tape Labels

If STANDARD labels are specified in the File Description entry, certain items within the label are automatically processed by the COBOL compiler. The remaining items may be used by the programmer by using the BEGINNING-LABEL and/or the ENDING-LABEL options of the LABEL RECORDS clause in the File Description entry.

For details concerning the form of the standard tape labels, see the publication, *IBM 1410/7010 Operating System; Basic Input/Output Control System*, Form C28-0322.

Record Formats for Tape Files

The data record formats that can be handled by the 1410/7010 COBOL compiler for files assigned to tape are:

1. Fixed-length, unblocked records with or without terminal record marks (Figure 4).
2. Variable-length, unblocked records with terminal record marks and without length checking (Figure 5).
3. Fixed-length, blocked records with terminal record marks (Figure 6).
4. Variable-length, unblocked records containing a Record Character-Count field and with or without terminal record marks (Figure 7).

The Record Character-Count field is a four-position field at the beginning of each record. It contains a count of the total number of characters in that record, including itself and the terminal record mark, if present.

5. Variable-length, blocked records with a Block Character-Count field and containing Record Character-Count fields. Terminal record marks are required (Figure 8).

A four-character Block Character-Count (BCC) field at the beginning of each block contains a count of the total number of characters in the block (including the four-character Block Character-Count field itself).

This count is used to check and correct wrong-length-record conditions. The BCC field must have AB zone bits (12-punch) over the units position.

This field is not a part of a record and therefore is not defined in a Record Description entry.

A Record Character-Count (RCC) field of one to four characters in each record contains a count of the total number of characters in that record, including itself and the terminal record mark. This field must be in the same relative position in each record (the number of characters in each "C1" in Figure 8 is the same), and must be the same length in each record of a given file. The "C2" fields in Figure 8 are all equal in length.

Record Formats for Unit-Record Files

CARD READ PUNCH RECORDS

Records of files assigned to the card reader or card punch must be 80 characters in length, unblocked, and may or may not have record marks in the 80th character position. In addition, these files must be in Move mode and even parity.

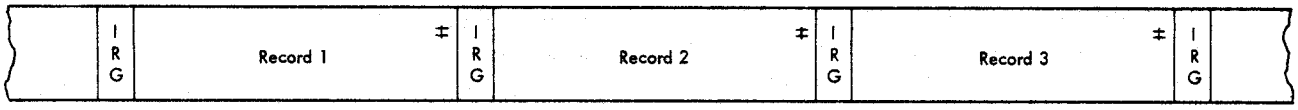


Figure 4. Fixed-Length, Unblocked Records

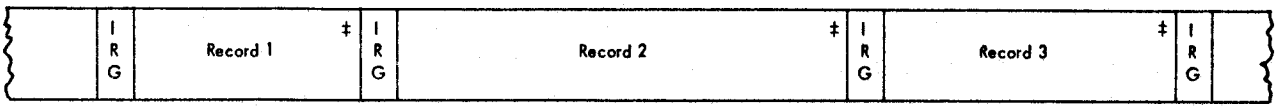


Figure 5. Variable-Length, Unblocked Records Without Length Checking

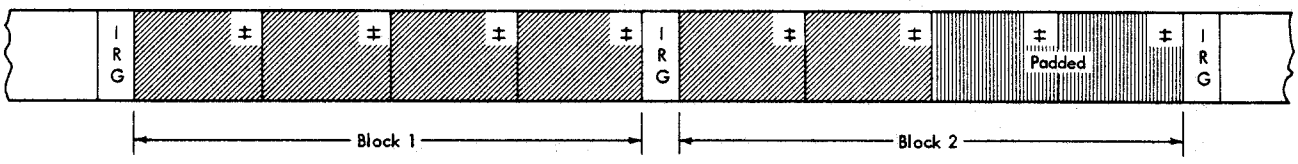


Figure 6. Fixed-Length, Blocked Records

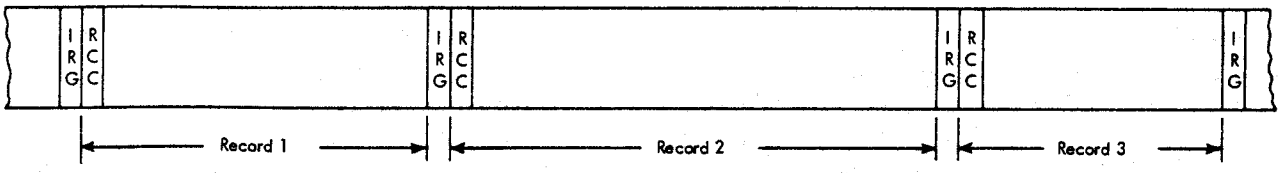


Figure 7. Variable-Length, Unblocked Records

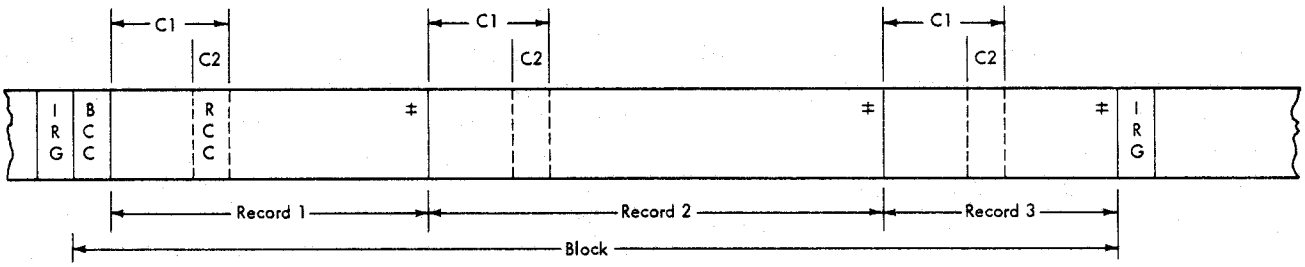


Figure 8. Variable-Length, Blocked Records

PRINTER RECORDS

Records of files assigned to the printer must be 132 characters, fixed-length, and unblocked. Files assigned to the printer must be in Move mode and even parity.

File Section

File Description Entry

A File Description entry must describe each file to be processed by the object program. It includes specifications for the mode in which the file is recorded, record and block size, label record information, and the names of the data records that make up the file.

The form of the File Description entry is:

```
FD file-name [ RECORDING MODE IS
                    { MOVE | LOAD } MODE { EVEN | ODD } PARITY ]
  [ BLOCK CONTAINS integer-1 { RECORDS | CHARACTERS } ]
RECORD CONTAINS [ integer-2 TO ]
                integer-3 CHARACTERS
                [ DEPENDING ON data-name-1 ]
LABEL RECORD[S] { ARE | IS }
    { STANDARD     [ WITH integer-4 CHARACTERS ]
                   [ BEGINNING-LABEL ]
                   [ ENDING-LABEL ]
    { OMITTED
    { NON-STANDARD [ WITH integer-4 CHARACTERS ]
                   [ BEGINNING-LABEL ]
                   [ ENDING-LABEL ]
[ VALUE OF FILE-IDENTIFICATION IS literal-1
                [ RETENTION-PERIOD IS integer-5 ] ]
DATA RECORD[S] { ARE | IS } data-name-2 [ data-name-3 ... ]
```

Level Indicator: The level indicator FD identifies the beginning of the File Description entry and precedes the file-name assigned by the programmer (Figure 9).

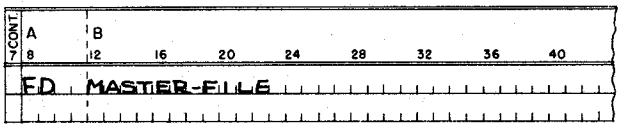


Figure 9. FD File-Name

RECORDING MODE Option: This option specifies the mode in which the file is recorded. (See the "Recording Modes" section of this publication.) If the RECORDING MODE option is omitted in the source program, the compiler assumes Move mode and even parity.

BLOCK CONTAINS Option: In addition to the details specified in the General Information Manual, the following information pertains to the BLOCK CONTAINS option.

If the file-name in the FD entry contains variable-length records, this entry must take the form:

BLOCK CONTAINS *integer-1* CHARACTERS

where integer-1 must be equal to or greater than the number of characters contained in the longest block of the file. This number includes the four-character Block Character-Count (BCC) field (see variable-length, blocked records in the section, "Record Formats for Tape Files").

RECORD CONTAINS Clause: This required clause is used to specify the size of a record in terms of the number of characters it contains and to indicate the record form. Integer-2 is used to specify the minimum number of characters in the smallest record of the file, whereas integer-3 indicates the maximum number of characters in the largest record. If all records in the file are exactly the same size, only integer-3 should be specified.

The DEPENDING ON data-name-1 option is required only when specifying variable-length records with Record Character-Count (RCC) fields. Data-name-1 is the name of the RCC field. The contents of this field indicate the number of characters in the record.

The following examples illustrate the use of the BLOCK CONTAINS option and the RECORD CONTAINS clause to specify each of the five record forms:

For fixed-length, unblocked records:

```
RECORD CONTAINS 80 CHARACTERS
```

For variable-length, unblocked records without length checking:

```
RECORD CONTAINS 100 TO 200 CHARACTERS
```

For fixed-length, blocked records:

```
BLOCK CONTAINS 5 RECORDS
RECORD CONTAINS 80 CHARACTERS
```

NOTE: Occasionally a fixed-length, blocked record file contains a partially completed block at the end of the file. If such a block appears, it will contain two types of records: data records and padding records. The user must provide for the processing of all padding records which appear in this block. The READ verb AT END branch will be taken when an attempt is made to read the next block, not when an attempt is made to read padding records from a partially completed block.

For variable-length, unblocked records:

```
RECORD CONTAINS 100 TO 200 CHARACTERS DE-
  PENDING ON RCC
```

For variable-length, blocked records:

```
BLOCK CONTAINS 504 CHARACTERS
RECORD CONTAINS 30 TO 50 CHARACTERS DE-
  PENDING ON RCC
```

When a 01 or a 77 level entry is allocated storage by the compiler, an additional storage position containing a group mark/ word mark is automatically generated at the end of the record. The diagnostic "Incompatible Record Size" is supressed if there is only one character deviation in record size. This is to allow for the possible use of a PICTURE symbol "K" (group mark/ word mark) which may be defined in the COBOL source program.

LABEL RECORD Clause: This clause is required in every File Description entry. For unit-record files, this clause must specify that label records are OMITTED. If STANDARD labels are specified for tape files, the file identification, the reel sequence, and the retention period are automatically checked.

If either STANDARD or NON-STANDARD is specified and the WITH integer-4 CHARACTERS option is desired, integer-4 must be 80 or 120. This is required in order to conform with the 1410 80-character and IBM Standard 120-Character tape labels. (For details concerning these labels see the publication, *Basic Input/Output Control System.*) If this option is not used, the label record size is assumed to be 120 characters.

NOTE: Actual size of nonstandard labels need not be exactly 80 or 120 characters, but may not exceed 120.

When a file contains standard tape labels, and no processing beyond that supplied by the compiler is required, STANDARD must be specified.

If additional processing of the standard tape label is desired, the programmer must specify STANDARD with BEGINNING-LABEL and/or ENDING-LABEL in conjunction with the USE verb. If either or both of these options are used, a Record Description entry that defines the entire label must be provided.

Example:

```
.
.
.
LABEL RECORDS ARE STANDARD BEGINNING-LABEL
ENDING-LABEL
.
.
.
01 BEGINNING-LABEL.
     02 . . .
     02 . . .
01 ENDING-LABEL.
     02 . . .
     02 . . .
```

When a file contains nonstandard labels and label processing is not desired, NON-STANDARD must be specified. Use of NON-STANDARD without additional options will cause the nonstandard labels to be bypassed in the object program.

Special processing of nonstandard labels can be accomplished by defining the label format with the BEGINNING-LABEL and ENDING-LABEL options in conjunction with the USE verb. No automatic testing takes place if NON-STANDARD is specified.

VALUE Option: The function of the VALUE option in the File Description entry is to specify the contents of data items in the label record of the file. The following two forms of the VALUE option are permitted for standard tape labels:

Form 1.

```
VALUE OF FILE-IDENTIFICATION IS literal-1
```

This form applies to both input and output files and is required if standard tape labels are used. Literal-1 must be a ten-character non-numeric literal.

Form 2.

```
VALUE OF FILE-IDENTIFICATION IS literal-1
RETENTION-PERIOD IS integer-5
```

This form applies to output files and must be supplied for each output file if standard tape labels are used. Integer-5 must be an integer (up to four digits with 120-character labels, and up to three digits with 80-character labels) indicating the number of days beyond the creation date the file is to be preserved. For files that are to be preservd indefinitely, the programmer inserts the digits "99" in the two high-order positions of the creation date (see "Standard Tape Labels").

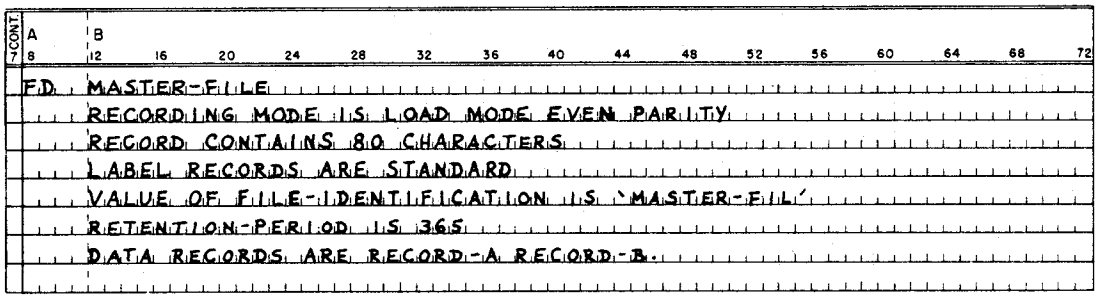


Figure 10. File Description Entry

If the appropriate VALUE option is not supplied when STANDARD labels are specified, a diagnostic message will not appear, but the results at execution time will be unpredictable.

DATA RECORD Clause: This clause is required in every File Description entry. Data-name-2, data-name-3, . . . etc., must each be the subject of a Record Description entry that has a level number of 01. The data-name order is not significant to the processor.

The appearance of more than one data-name in this clause means that the file contains a corresponding number of different types of data records. These records may be of different sizes and formats.

Implicit redefinition caused by multiple records in the DATA RECORD clause has the same effect as a REDEFINES clause. Therefore, a MOVE CORRESPONDING can not be used for either implicit or explicit redefinition of the receiving field.

Figure 10 illustrates a sample File Description entry.

Record Description Entry

A Record Description entry specifies to the compiler the characteristics of each item of a data record. Every item given a separate name must be described in a separate entry in the same order in which it appears in the record. Each Record Description entry consists of a level-number, a data-name, and a series of independent clauses. The form of a Record Description is:

level-number { *data-name* / FILLER } [REDEFINES . . .]
[SIZE . . .] [CLASS . . .] [USAGE . . .]
[OCCURS . . .] [POINT . . .] [SIGNED . . .]
[VALUE . . .] [PICTURE . . .] [BLANK WHEN ZERO].

Level-Number: The level-number shows the relationship between items in a record. Each level-number must be associated with a data-name or with the key word FILLER, as shown in the following general format:

level-number { *data-name* / FILLER }

A detailed description of level-numbers can be found in the General Information Manual.

REDEFINES Clause: The general form of the REDEFINES clause is:

level-number data-name-1 [REDEFINES *data-name-2*]

The size of data-name-1 and data-name-2 should be equal. Data-name-2 must not be subscripted.

There are two types of COBOL redefinition: implici and explicit. Implicit redefinition is caused by multiple records named in the DATA RECORDS clause. All of these records implicitly share the same work area. Explicit redefinition is caused by the use of the REDEFINES clause following a data-name in the Data Division. In this case, both the redefining and the redefined data-name share the same work area by explicit definition. Incompatable REDEFINES clauses are flagged at compilation time.

See the General Information Manual for details concerning the use of the REDEFINES clause. Additional information, pertinent to the use of REDEFINES in programs that are to be compiled on systems other than the 1410/7010 appears in the "Compatibility Considerations" section of this manual.

SIZE Clause: The general form of the SIZE clause is:

[SIZE IS *integer-1* [{ CHARACTER[S] / DIGIT[S] }]]

See the General Information Manual for details concerning the use of this clause.

CLASS Clause: In addition to the details in the General Information Manual concerning the use of this clause, the reader should note that if a CLASS statement is omitted for a data-item and the USAGE clause specifies COMPUTATIONAL, numeric class is implied. In the absence of any CLASS specification or implication, alphanumeric class is assumed. Alphameric class is always assumed for a group item. Numeric class items must not exceed 18 digits. For report items, the number of numeric characters represented must not exceed 18.

The general form of the CLASS clause is:

[CLASS IS { ALPHABETIC / NUMERIC / ALPHANUMERIC / AN }]

USAGE Clause: The USAGE clause does not in any way affect the internal representation of data in the IBM 1410/7010 Data Processing Systems. All data is represented internally in BCD (binary-coded decimal)

form and no distinction is made between COMPUTATIONAL and DISPLAY usage. If the USAGE clause for a data-item specifies COMPUTATIONAL and a CLASS statement is omitted, the class is assumed to be numeric. The general form of the USAGE clause is:

$$\left[\text{USAGE IS } \left\{\begin{matrix}\underline{\text{COMPUTATIONAL}}\\ \underline{\text{DISPLAY}}\end{matrix}\right\}\right]$$

OCCURS Clause: The general form of the OCCURS clause is:

$$\left[\underline{\text{OCCURS}}\ \textit{integer-2}\ \text{TIME[S]}\right]$$

See the General Information Manual for details concerning the use of this clause.

POINT Clause: The general form of the POINT clause is:

$$\left[\underline{\text{POINT}}\ \text{LOCATION IS } \left\{\begin{matrix}\underline{\text{LEFT}}\\ \underline{\text{RIGHT}}\end{matrix}\right\}\ \textit{integer-3}\ \text{PLACE[S]}\right]$$

See the General Information Manual for details concerning the use of this clause.

SIGNED Clause: A numeric data item will have an operational sign if this clause is used. An operational sign should be specified for the result field of any arithmetic statement where the sign is a consideration. Additional details concerning the use of the SIGNED clause are found in the General Information Manual. The general form of this clause is:

$$\left[\underline{\text{SIGNED}}\right]$$

VALUE Clause: The general form of the VALUE clause is:

$$\left[\underline{\text{VALUE}}\ \text{IS}\ \textit{literal}\right]$$

In addition to the details specified in the General Information Manual, the following information pertains to the use of this clause:

1. If the VALUE clause specifies a numeric literal with a preceding sign, the operational sign is created only if the programmer specifies the PICTURE symbol "S" or the SIGNED clause.
2. The VALUE clause can only be used to refer to elementary items.
3. The VALUE clause has no meaning for report items, and cannot be used to specify their initial values.
4. Neither a record mark (≡) nor a group mark (≣) can be used within the VALUE clause (see PICTURE symbols "J" and "K").

PICTURE Clause: The general form of the PICTURE clause is:

$$\left[\underline{\text{PICTURE}}\ \text{IS}\ \begin{matrix}\textit{any allowable combination of}\\ \textit{characters and symbols}\end{matrix}\right]$$

The PICTURE clause can only be used to describe elementary items. It is recommended that, wherever possible, the programmer use this clause instead of the SIZE, POINT, CLASS, and BLANK clauses of a Record Description entry. The PICTURE clause specifies the characteristics of an elementary item in a more compact form, and can therefore be processed more efficiently.

Non-computational numeric data-names greater than 18 digits in length should be declared alphameric. This results in more efficient processing at compilation time and at object program execution time.

Non-computational numeric data-names must not exceed 99 digits. Data names containing more than 99 positions must be declared as alphabetic or alphameric, or an addressing error will occur.

In addition to the rules given in the General Information Manual for forming a picture of a data item, the following information pertains to the use of the PICTURE clause:

1. The only way to define a record mark or group mark is by using a PICTURE symbol. The special PICTURE symbol "J" is used to indicate a one-character field containing a record mark (≡), and the special PICTURE symbol "K" is used to indicate a one-character field containing a group mark (≣). When used, the PICTURE symbol "J" or "K" must be the only character in the PICTURE.
2. The PICTURE symbol "S" is used to indicate an operational sign (see the SIGNED clause).
3. For report items, the maximum number of characters that can be represented by a PICTURE is 99.
4. The PICTURE symbol "V" to the right or left of PICTURE symbol "P" is redundant and invalid.
5. PICTURE symbol "Z" may appear to the right of a decimal point in a PICTURE only if all numeric character positions are represented by "Z"s. The same rule applies to the replacement characters "*", "+", and "−".

BLANK WHEN ZERO Clause: The general form of the BLANK WHEN ZERO clause is:

$$\left[\underline{\text{BLANK}}\ \text{WHEN}\ \underline{\text{ZERO}}\right]$$

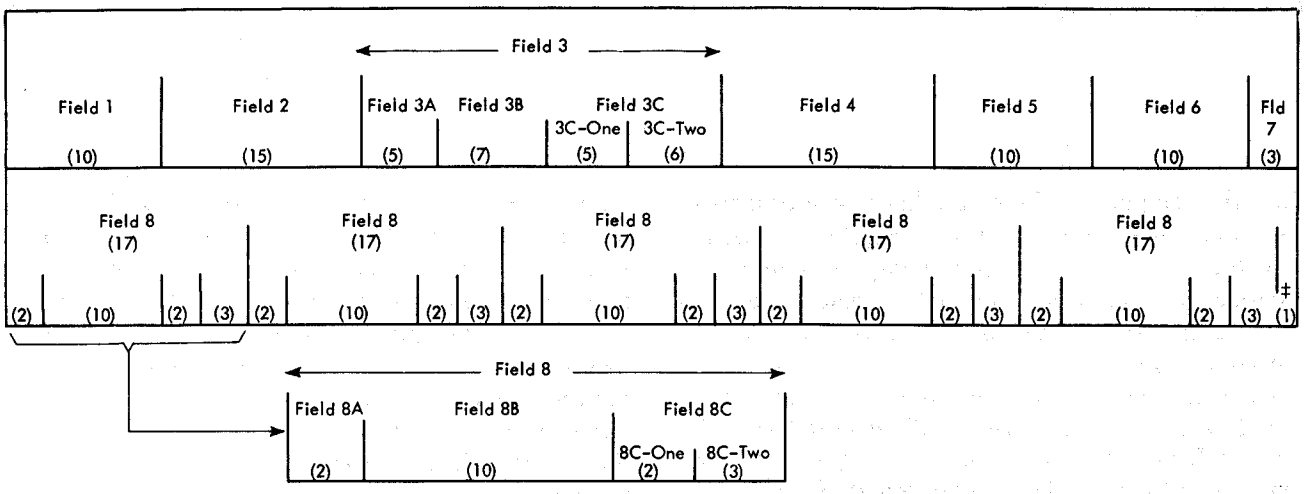
See the General Information Manual for details concerning the use of this clause.

Figure 11 illustrates a sample Record Description entry.

Working-Storage and Constant Sections

The Record Description entries described for the File Section apply also to the Working-Storage and Constant Sections. These sections begin with the header line "WORKING-STORAGE SECTION," or "CONSTANT SECTION," and are followed immediately by the Record Description entries.

In addition to the details specified in the General Information Manual, the following considerations pertain to the Working-Storage and Constant Sections.




```
SERIAL  CONT  A    B
4  6    7    8    12   16   20   24   28   32   36   40   44   48   52   56   60   64   68   72
010     01   MASTER-INPUT-RECORD.
020          02 FIELD1 PICTURE IS A(10).
030          02 FIELD2 PICTURE IS A(15).
040          02 FIELD3.
050             03 FIELD3A PICTURE IS X(5).
060             03 FIELD3B PICTURE IS X(7).
070             03 FIELD3C.
080                04 FIELD3C-ONE PICTURE IS 999V99.
090                04 FIELD3C-TWO PICTURE IS 99999V9.
100          02 FIELD4 PICTURE IS A(15).
110          02 FIELD5 PICTURE IS X(10).
120          02 FIELD6 PICTURE IS 9(8)V99.
130          02 FIELD7 PICTURE IS 999.
140             88 TYPE1 VALUE IS 001.
150             88 TYPE2 VALUE IS 359.
160             88 TYPE3 VALUE IS 751.
170          02 FIELD8 OCCURS 5 TIMES.
180             03 FIELD8A PICTURE IS 99.
190             03 FIELD8B PICTURE IS 9(10).
200             03 FIELD8C.
210                04 FIELD8C-ONE PICTURE IS 99.
220                04 FIELD8C-TWO PICTURE IS 999.
230          02 RECORD-MARK SIZE IS 1 CLASS IS AN.
```

Figure 11. Record Description Entry

If the VALUE clause is not used to define the initial values of Working-Storage items, these values will be unpredictable.

Constant Section elementary items must include a VALUE clause or one of the PICTURE symbols "J" and "K", unless associated with a REDEFINES clause.

Tables of constants may be formed using the OCCURS clause and may be referenced by subscripts. The procedure for constructing tables of constants is described fully in the General Information Manual.

Added Features of the Data Division

An optional feature, not specified in the General Information Manual, but contained in the 1410/7010 COBOL language is:

The DEPENDING ON option of the RECORD CONTAINS clause.

Once the computer and the data have been described, the programmer gives the COBOL compiler the instructions that specify the data processing steps the object program is to perform. Information that directs the compiler is also supplied by the programmer in the Procedure Division.

The COBOL verbs are the main elements in the Procedure Division and they are described in detail in the General Information Manual. However, for the convenience of the user, the general format of each COBOL verb is shown in this publication.

Information not included in the General Information Manual, but applicable to verbs which have additional meaning when used in a 1410/7010 COBOL source program, is supplied in later sections of this manual.

Compiler Directing Declaratives

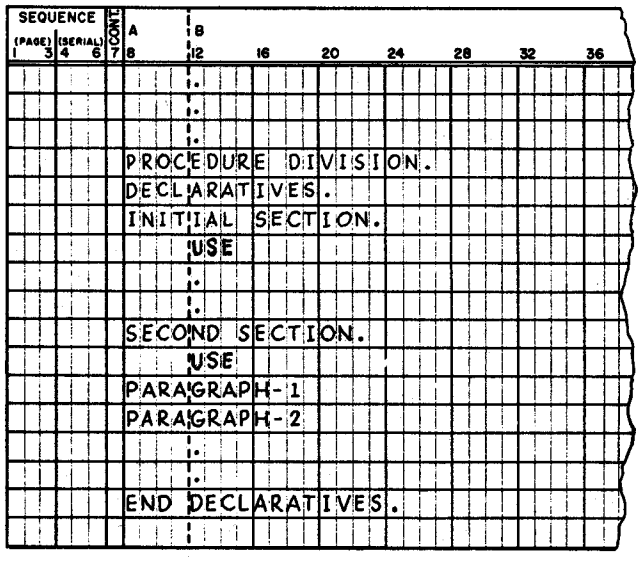
Declaratives are compiler-directing statements that operate under the control of the "main body" of the Procedure Division or the Input/Output Control System. Declaratives consist of the USE verb and any associated procedures. If present, Declaratives:

1. must be grouped at the beginning of the Procedure Division, and
2. must be preceded by the key word DECLARATIVES and followed by the key words END DECLARATIVES, and
3. must be included in the first phase of a multiphase COBOL program that is to be executed from the SOF.
4. are an isolated part of the program. No reference to procedure-names may be made between the Declaratives and the remaining part of the Procedure Division.

A section-name in the declarative portion of the Procedure Division must be followed by a USE statement, and the USE statement must be followed by a paragraph-name. A section-name in the nondeclarative portion of the Procedure Division must be followed by a paragraph-name.

USE Verb

The USE verb is used to specify procedures for handling input/output errors and label processing (see "LABEL RECORD" clause) in addition to the procedures normally provided by the COBOL compiler. Each occurrence of the USE verb constitutes a complete section of the Procedure Division, and must therefore be preceded by a section-name. The remainder of the section consists of one or more paragraphs specifying the procedures to be used (Figure 12). A "USE" section must follow the rules for a section which is "performed" (see the PERFORM verb in the General Information Manual for details). For example, if the logic of a procedure requires a conditional exit prior to the final statement, the EXIT verb must be used. The section-name of a USE section must not be referenced by an ALTER, GO TO, or PERFORM statement since the USE entry is appended to this section header. The paragraph-name of the last paragraph in a USE section must not be referenced by an ALTER or PERFORM statement since the USE exit is appended to this paragraph.



• Figure 12. Declaratives

The following options of the USE verb are available.

Option 1

section-name SECTION. USE AFTER STANDARD ERROR PROCEDURE ON *file-name-1* [*file-name-2* . . .]
paragraph-name. (any COBOL statements including DISPLAY using the console printer but excluding all other input/output statements)

This option allows the programmer to specify additional procedures if standard error-correction procedures do not suffice. Standard error procedures attempt to correct the error in accordance with 1410/7010 IOCS standards. If the error persists, the error condition is ignored and processing continues. Thus, the programmer has the following options: ignore the error condition and process the record, or by means of the USE verb perform special processing for error records (e.g., set a switch, display a console message, etc.).

Option 2

section-name SECTION. USE $\left\{\begin{matrix}\text{BEFORE}\\\text{AFTER}\end{matrix}\right\}$ STANDARD $\left[\left\{\begin{matrix}\text{BEGINNING}\\\text{ENDING}\end{matrix}\right\}\right]\left[\left\{\begin{matrix}\text{REEL}\\\text{FILE}\end{matrix}\right\}\right]$ LABEL PROCEDURE ON *file-name.*

paragraph-name. (any COBOL statements including DISPLAY using the console printer but excluding all other input/output statements)

This option is used to:

1. Perform processing of standard tape labels beyond that supplied by the COBOL compiler.
2. Perform any desired processing of nonstandard labels.

If both BEGINNING and ENDING are omitted, the designated procedures will be executed for both beginning (header) and ending (trailer) labels.

If both REEL and FILE are omitted, the designated procedures will be executed upon detection of both end-of-reel and end-of-file conditions.

Eight label exits per FD are provided by the COBOL compiler:

Input Tapes

Label	*Option*	*Exit*
BEGINNING	BEFORE	B
BEGINNING	AFTER	C
ENDING	BEFORE	E
ENDING	AFTER	F

Output Tapes

Label	*Option*	*Exit*
BEGINNING	BEFORE	K
BEGINNING	AFTER	L
ENDING	BEFORE	P
ENDING	AFTER	Q

For further information, see *IBM 1410/7010 Operating System; Basic Input/Output Control System* Figure 65.

The same exits are used for the REEL option, the FILE option, or both of these options (see the section on the extended use of the IOCS in the *Basic Input/Output Control System* publication). Therefore, if multiple USE statements referencing the same FD are used, and if the exits conflict, the REEL and FILE options of the USE verb should not be included. In this case, the programmer's USE section procedural statements should determine if the condition is REEL or FILE.

If the RERUN clause or Option 2 of the USE verb is used: (1) an MDM file and IOKDF label option D for both 80- and 120-character labels must be included at System Generation, and (2) the file associated with the RERUN clause or Option 2 of the USE verb must be a labeled file. Failure to include these items at System Generation may cause the system symbols /LRA/ and /LRC/ to remain undefined when the COBOL object program is loaded by the Linkage Loader.

If neither the RERUN clause nor Option 2 of the USE verb is used, the undefined system symbols /LRA/ and /LRC/ will not affect the execution of the object program.

Input/Output Verbs

OPEN and CLOSE

The COBOL compiler provides the facility for opening an input or output file, processing it, closing it, and subsequently reopening it as an input or output file.

The OPEN verb is used to initiate the processing of one or more input and/or output files. The format of the OPEN verb is:

OPEN $\left[\text{INPUT}\ \textit{file-name-1}\left[\textit{file-name-2}\ldots\right]\right]$ $\left[\text{OUTPUT}\ \textit{file-name-3}\left[\textit{file-name-4}\ldots\right]\right]$

The CLOSE verb is used to terminate processing of one or more input and/or output reels or files. Provision for optionally locking or not rewinding is also included. The format of the CLOSE verb is:

CLOSE *file-name-1* $\left[\text{REEL}\right]\left[\text{WITH}\left\{\begin{matrix}\text{LOCK}\\\text{NO REWIND}\end{matrix}\right\}\right]$ $\left[\textit{file-name-2}\ldots\right]$

NOTE: The reel option must not be used if more than one file appears on a reel and the remaining files are to be processed, or if the file consists of a single physical reel. If used, reels after the first will be rewound.

See the General Information Manual for details concerning the OPEN and CLOSE verbs.

READ

The function of this verb is to make the next record from an input file available for processing. The general form of the READ verb is:

READ *file-name* RECORD $\left[\text{INTO}\ \textit{area-name}\right]$ AT END *any imperative statement*

In addition to the details specified in the General Information Manual, the following considerations pertain to the use of the READ verb:

1. An OPEN statement for the file must be executed prior to the execution of the first READ for that file.

2. When a READ is executed, the next record of the file becomes accessible in the input area defined by the associated Record Description entry in the File Section of the Data Division. The record remains available in the input area until the next READ (for that file) is executed. The named file must be defined by an FD entry in the Data Division of the program.

3. Every READ statement must include an AT END clause containing any imperative statements; i.e., any single verb with its operand(s), or a sequence of verbs with their operands terminated by a period and con-

taining no explicit or implied conditional expressions. Once an AT END statement has been executed, any attempt to READ from the file will constitute an error unless a subsequent CLOSE and OPEN have been executed.

NOTE: When reading a file containing fixed-length, blocked records, the end-of-file condition does not necessarily occur following the last logical record. Therefore, the programmer must test for a record consisting of all padding characters to ensure detection of the end of the logical file.

4. The INTO area-name option converts the READ into a READ and MOVE. The area-name specified must be the name of either a Working-Storage record area or an output record area. When this option is used, the current record becomes available in the input area, as well as in the area specified by area-name. If the format of the INTO area differs from that of the input record, the data will be moved in accordance with the rules for the MOVE verb without the CORRESPONDING option. It will be assumed that the area specified by area-name will be completely filled by information from the input record. If this is not the case, READ and MOVE should be used rather than READ INTO.

5. Each time an end-of-reel condition occurs in a reel other than the last, the READ verb causes the following operations to take place:

a. If labels are present (as specified in the FD for that file) the standard end-of-reel label subroutine of the Input/Output Control System is executed.
b. A tape alternation occurs, if appropriate.
c. If labels are present, the standard beginning-of-reel label subroutine is executed.
d. If RERUN has been specified for this file, a checkpoint record is written.
e. The next record in the file is made available for processing.

WRITE

The purpose of the WRITE verb is to release a record for insertion in an output file. The format of a WRITE statement is:

WRITE *record-name* [FROM *area-name*]

In addition to the details specified in the General Information Manual, the following considerations pertain to the use of the WRITE verb:

1. If the user desires to write records which have been described by the RENAMING option (see the "FILE CONTROL" paragraph), the record-name must always be qualified by the file-name.

2. If the FROM option is used, information will be transmitted from area-name with or without word marks, depending upon the RECORDING MODE of the file associated with record-name. If the file is defined in the Load mode, word marks will be transmitted. If the file is defined in the Move mode, word marks will not be transmitted. Area-name must be the name of an input record or a Working-Storage or Constant Section record area.

DISPLAY

The format of the DISPLAY verb is:

DISPLAY { *data-name-1* / *literal-1* } [{ *data-name-2* / *literal-2* } ...]
[UPON *mnemonic-name*]

In addition to the details specified in the General Information Manual, the following information pertains to the use of the DISPLAY verb:

1. DISPLAY literals must be non-numeric.

2. The Operating System's Standard Punch Unit and Standard Print Unit may be equated with mnemonic-names in the SPECIAL-NAMES paragraph of the Environment Division. If the UPON option is omitted, the console printer will be used as the standard Display Device.

3. Depending on the 1403 printing chain or the console printer type head, certain characters will not be displayed. See the publication, *IBM 1410 Principles of Operation*, Form A22-0526 or *IBM 7010 Principles of Operation*, Form A22-6726, for further details.

4. If a printer is used, it will be assumed that the carriage tape has a channel-1 punch.

5. Information of any length can be displayed on any display device.

6. A standard set of error procedures is provided by the resident Input/Output Control System for use in the execution of the DISPLAY verb.

Figure 13 shows a DISPLAY statement that will cause the contents of the field GRAND-TOTAL to be typed on the console printer when the object program is executed.

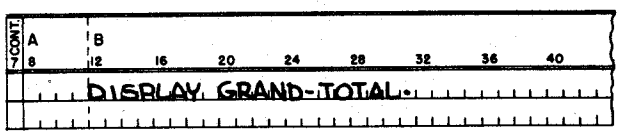


Figure 13. Standard DISPLAY Device

Figure 14 shows a DISPLAY statement that will cause the contents of the field GRAND-TOTAL to be displayed in 80-character records on the Standard Punch Unit when the object program is executed, assuming that the mnemonic-name PUNCH has been equated with the SYSTEM-OUTPUT-PUNCH.

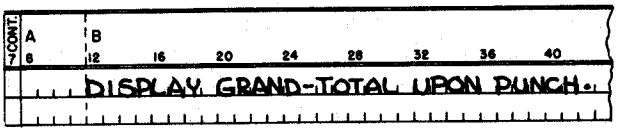


Figure 14. Punch DISPLAY

ACCEPT

The function of the ACCEPT verb is to obtain low-volume data from the Operating Systems Standard Input Unit. The Standard Input Unit is the only device from which information can be accepted. The general form of the ACCEPT verb is:

ACCEPT *data-name*

Figure 15 shows an ACCEPT statement that will cause data to be read from the Standard Input Unit and moved into the area defined by the data-name CANCELLATIONS. If this area contains more than 80 characters, sufficient card images will be read to fill it.

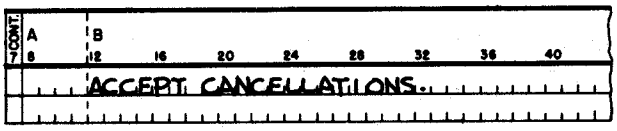


Figure 15. ACCEPT Verb

A standard set of error procedures is provided by the resident Input/Output Control System for use in the execution of the ACCEPT verb.

Data Manipulation Verbs

MOVE

The MOVE verb can be used in either of two formats:

Option 1

MOVE { *data-name-1* / *literal* } TO *data-name-2* [*data-name-3* ...]

Option 2

MOVE CORRESPONDING *data-name-1* TO *data-name-2* [*data-name-3* ...]

In addition to the details specified in the General Information Manual, the following considerations pertain to the use of Options 1 and 2 of the MOVE verb:

1. The following moves are aligned by decimal point:
   a. Elementary numeric to elementary numeric.
   b. Elementary numeric to elementary alphanumeric report.
   c. Elementary non-numeric to elementary numeric. (The elementary non-numeric item is assumed to be an integer.)
2. All other moves are left-justified.
3. When figurative constants are used as operands of the MOVE verb, the size of the receiving area determines the number of characters that are moved. For example, if the size of AREA-A in Figure 16 is five positions, its value after execution is five nines (99999). If the receiving field is a report item, no editing will take place. Instead, the entire field will be filled with the constant being moved.

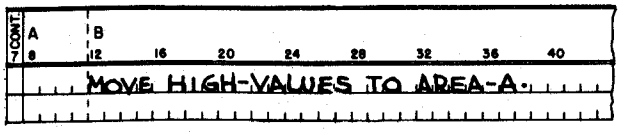


Figure 16. MOVE Verb

4. The CORRESPONDING option will not match a data-name which is redefined at a lower level than that of the operand in the CORRESPONDING statement.

EXAMINE

The general form of the EXAMINE verb is:

EXAMINE *data-name*

{ TALLYING { ALL / LEADING / UNTIL FIRST } *literal-1* [REPLACING BY *literal-2*] }
{ REPLACING { ALL / LEADING / [UNTIL] FIRST } *literal-3* BY *literal-4* }

See the General Information Manual for details concerning the EXAMINE verb.

Arithmetic Verbs

The following rules apply to the arithmetic verbs:

1. All data-names used in arithmetic statements must represent elementary numeric data items that are defined in the Data Division of the program. A data-name that is defined within the Constant Section cannot appear as the result field of an arithmetic statement.
2. All literals used in arithmetic statements must be numeric.
3. The maximum size of any operand (data-name or literal) is 18 digits. If the format for any operand specifies a size greater than 18 digits, the compiler will produce an error message when it encounters the discrepancy.
4. Intermediate result fields generated for the evaluation of arithmetic expressions (formulas) will always have a PICTURE of S9(10)V9(10). If greater precision is desired, the simple arithmetic verbs (i.e., ADD, SUBTRACT, MULTIPLY, and DIVIDE) must be used.
5. Decimal-point alignment is supplied automatically throughout computations.
6. The format of any data item involved in computations (e.g., addends, subtrahends, multipliers, etc.) cannot contain editing symbols. If this rule is violated, the compiler will indicate the error by an appropriate message. Operational signs and implied decimal points are not considered editing symbols. The data-name in the GIVING option and the result field in the COMPUTE verb format represent data items which must not enter into computations if they contain editing symbols.

7. The only figurative constant permitted in arithmetic statements is ZERO (or ZEROS and ZEROES).

8. For the simple arithmetic verbs the data characteristics of the receiving field control the precision of the operation; therefore, no high-order digit will be lost without creating the SIZE ERROR condition. All specified decimal positions will also be produced.

9. For use with the SIZE ERROR option "any imperative statement" is any single verb with its operand(s) or a sequence of verbs with their operands terminated by a period and containing no explicit or implied conditional expressions.

10. If exponentiation is used in a COMPUTE expression, the exponent must be an integer. Negative exponents are permitted. Additional information on each of the arithmetic verbs may be found in the General Information Manual.

ADD

The general form of the ADD verb is:

```
ADD { data-name-1 } [ { data-name-2 } ... ]
    { literal-1   }   { literal-2   }

[ { TO    } data-name-n ] [ ROUNDED ]
  { GIVING }

                [ ON SIZE ERROR any imperative statement ]
```

An ADD statement must name at least two addends. For additional details concerning the ADD verb, see the General Information Manual.

CORRESPONDING Option: The CORRESPONDING option of the ADD verb allows the programmer to specify the addition of corresponding items in one operation in a manner similar to MOVE CORRESPONDING.

The general form of ADD CORRESPONDING is:

```
ADD CORRESPONDING data-name-1 TO data-name-2
[ ROUNDED ]
                [ ON SIZE ERROR any imperative statement ]
```

Numeric elementary items within data-name-1 are added to numeric elementary items with matching names in data-name-2. Data-name-1 and data-name-2 must be nonelementary items. The rules stated for the simple ADD verb apply to each pair of items in the ADD CORRESPONDING option.

The rules concerning redefined areas are the same for the ADD CORRESPONDING option as for the MOVE CORRESPONDING option.

The ROUNDED option and the size ERROR option of the ADD verb may also be used with ADD CORRESPONDING. For a detailed description of these two options, see the General Information Manual.

NOTE: When SIZE ERROR is used in conjunction with CORRESPONDING, the SIZE ERROR test is made only after the completion of all the add operations. If any of the additions produced a SIZE ERROR, the resultant field for that add remains unchanged, and the "any imperative statement" is executed.

To illustrate the use of the ADD CORRESPONDING option, assume that the programmer wishes to add items from a work area named RECEIPTS to corresponding items in an area designated STOCK-ON-HAND. He would write this statement:

```
ADD CORRESPONDING RECEIPTS TO STOCK-ON-HAND
```

Figure 17 shows what will result from this statement. Note that noncorresponding items in the STOCK-ON-HAND area are not affected.

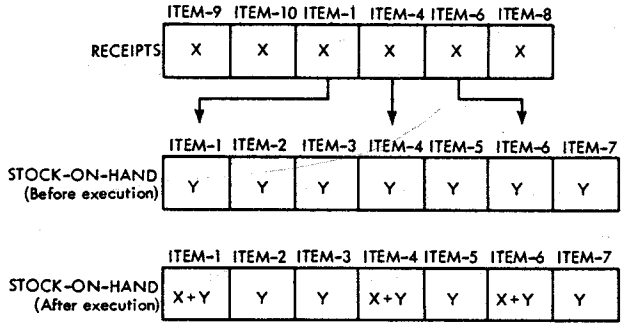


Figure 17. ADD CORRESPONDING

SUBTRACT

The general form of the SUBTRACT verb is:

```
SUBTRACT { data-name-1 } [ { data-name-2 } ... ]
         { literal-1   }   { literal-2   }

        FROM { data-name-n } [ GIVING data-name-m ]
             { literal-n   }

[ ROUNDED ]
                [ ON SIZE ERROR any imperative statement ]
```

A SUBTRACT statement must name at least one subtrahend and one minuend. For further details concerning the SUBTRACT verb, see the General Information Manual.

CORRESPONDING Option: The CORRESPONDING option of the SUBTRACT verb functions in the same way as the CORRESPONDING option of the ADD verb.

The general form of SUBTRACT CORRESPONDING is:

```
SUBTRACT CORRESPONDING data-name-1
                FROM data-name-2 [ ROUNDED ]
                [ ON SIZE ERROR any imperative statement ]
```

MULTIPLY

The general form of the MULTIPLY verb is:

```
MULTIPLY { data-name-1 } BY { data-name-2 }
         { literal-1   }    { literal-2   }
        [GIVING data-name-3] [ROUNDED]
        [ON SIZE ERROR any imperative statement]
```

DIVIDE

The general form of the DIVIDE verb is:

```
DIVIDE { data-name-1 } INTO { data-name-2 }
       { literal-1   }      { literal-2   }
      [GIVING data-name-3] [ROUNDED]
      [ON SIZE ERROR any imperative statement]
```

COMPUTE

The general form of the COMPUTE verb is:

```
COMPUTE data-name-1 [ROUNDED]
    = arithmetic expression
    [ON SIZE ERROR any imperative statement]
```

Procedure Branching Verbs

GO TO

There are two formats in which the GO TO verb can be used:

Option 1

```
GO TO [procedure-name]
```

Option 2

```
GO TO procedure-name-1 procedure-name-2
    [procedure-name-3 ...] DEPENDING ON data-name
```

For additional information concerning the GO TO verb, see the General Information Manual.

ALTER

The general form of the ALTER verb is:

```
ALTER procedure-name-1 TO PROCEED TO
      procedure-name-2 [procedure-name-3 TO
      PROCEED TO procedure-name-4 ...]
```

A GO TO sentence that is to be altered must be:

1. An unconditional GO TO sentence
2. Written as a separate paragraph consisting solely of the GO TO sentence, preceded by a procedure-name

PERFORM

There are five formats in which the PERFORM verb can be used. These are:

Option 1

```
PERFORM procedure-name-1 [THRU procedure-name-2]
```

Option 2

```
PERFORM procedure-name-1 [THRU procedure-name-2]
                         { integer-1   } TIME[S]
                         { data-name-1 }
```

Option 3

```
PERFORM procedure-name-1 [THRU procedure-name-2]
                         UNTIL condition-1
```

Option 4

```
PERFORM procedure-name-1 [THRU procedure-name-2]
                         VARYING data-name-1 FROM
                         { numeric-literal-1 } BY { numeric-literal-2 }
                         { data-name-2       }    { data-name-3       }
                         UNTIL condition-1
```

Option 5

```
PERFORM procedure-name-1 [THRU procedure-name-2]
                         VARYING subscript-name-1
                         FROM { integer-1   } BY { integer-2   }
                              { data-name-1 }    { data-name-2 }
                         UNTIL condition-1 [AFTER subscript-name-2
                         FROM { integer-3   } BY { integer-4   } UNTIL
                              { data-name-3 }    { data-name-4 }
                         condition-2] [AFTER subscript-name-3 FROM
{ integer-5   } BY { integer-6   } UNTIL condition-3]
{ data-name-5 }    { data-name-6 }
```

Compiler Directing Verbs

ENTER

The ENTER verb, used with the CALL verb, allows the programmer to construct multiphase COBOL programs and to incorporate into his object program FORTRAN and/or Autocoder compiled subprograms. The incorporation of subprograms is performed at the time the object program is processed by the Linkage Loader. (See the publication *IBM 1410/7010 Operating System; System Monitor,* Form C28-0319.) Each ENTER statement must constitute a separate paragraph in the source program. The form of the ENTER verb is:

```
ENTER { COMMUNICATION-MODE }
      { COBOL              }
```

The entry ENTER COMMUNICATION-MODE precedes the calling of the subprogram(s). The CALL verb specifies the subprogram(s) to be included in the object pro-

gram. The entry ENTER COBOL must terminate the list of subprograms. COMMUNICATION-MODE may be entered any number of times in a program.

The COMMUNICATION-MODE can be used to perform the following functions:

1. Read from disk storage devices directly into the WORKING STORAGE SECTION of the DATA DIVISION.
2. Write to disk storage devices directly from the WORKING STORAGE SECTION of the DATA DIVISION.
3. Share an on-line printer in an operating system that has full object-program carriage-control capabilities. This technique is usually faster than the DISPLAY verb method. The COBOL program must enter an Autocoder subprogram in order to use the Resident monitor Print Routine; the Autocoder subprogram must contain a specific linkage sequence. Use of the Resident Monitor Print routine, and the linkage sequence required for its execution, are described in the *System Monitor* publication.
4. Share an on-line card reader with the operating system (see "Read Routine" in *System Monitor*). This technique is usually faster than the ACCEPT verb technique.
5. Share an on-line card punch with the operating system (see "Punch Routine" in *System Monitor*). This technique is usually faster than the DISPLAY verb technique.

CALL

The general form of the CALL verb is:

CALL { *linkage-symbol* / *subprogram-name* / *system-symbol* } [USING { *file-name-1* / *data-name-1* / *literal-1* } [{ *file-name-2* / *data-name-2* / *literal-2* } ...]] . [CALL ...] .

Subprogram-1 is the name contained in the TITLE card of the subprogram. The CALL verb causes the COBOL compiler to generate an *imbedded* call for the named subprogram. When the imbedded call is processed by the Linkage Loader, it is converted into a branch to the first character of the called subprogram.

The USING option specifies the required parameters (data-names and/or literals) for the subprogram. These parameters reference data within the COBOL program and are the only means of communication between the main program and the subprogram. At object program run time, these parameters are represented by a sequence of five-character addresses of the appropriate data, with a word mark over the high-order position of each address. This list is followed by a terminal "No Operation" instruction. (The number of parameters is used by the called subprogram to determine the point at which control is to be returned to the main program.) Although any number of parameters may be specified, a maximum of two subscripted data-names may appear in a given USING option.

The format of the CALL statement used to call and enter a phase is

CALL CHAIN USING { *data-name-1* / *literal-1* } .

The above CALL statement, used with the ENTER verb, permits the CHAIN feature to be used with the Linkage Loader. See *System Monitor* for additional details of the CHAIN feature, including specifications for building multiphase COBOL object programs.

The COBOL compiler will generate an imbedded call for the CHAIN subprogram. This call will be followed by the address of the location of literal-1 or data-name-1. There is a word mark in the high-order position of the field with the address. The sequence is terminated with a NOP instruction.

Literal-1 or data-name-1 should be unsigned and three digits long. This literal or data-name represents the phase number field (columns 6-8) of a Linkage Loader PHASE card which will be generated by a Linkage card.

A CALL statement used to call and enter a phase may not appear within the DECLARATIVES portion of a program.

The CALL verb may be used only after the COMMUNICATION-MODE has been entered. No other verb may appear within the COMMUNICATION-MODE.

The following CALL statement will generate the illustrated calling sequence. For clarity, the calling sequence is presented in standard Autocoder format. However, the calling sequence is written on the Go file (or punched) in standard 1410/7010 relocatable format.

CALL [SUBPROGRAM] [USING
[*file-name-1, ... file-name-n*]
[*data-name-1, ... data-name-n*]
[*literal-1 , ... literal-n*]]

CALLING SEQUENCE GENERATED

OPERATION CODE	OPERAND	CHARACTER COUNT
DCWS	SUBPROGRAM	7
DCW	FILE-NAME-1	5
.	.	.
.	.	.
.	.	.
DCW	FILE-NAME-N	5
DCW	DATA-NAME-1	5
.	.	.
.	.	.
.	.	.
DCW	DATA-NAME-N	5
DCW	LITERAL-1	5
.	.	.
.	.	.
.	.	.
DCW	LITERAL-N	5
NOP	.	1

The DCW address constant of FILE-NAME-N points at the low-order character of the File List Origin field of the File Table.

The DCW address constant of DATA-NAME-N points at the low-order character of the data field. A word mark may appear over the high-order character of the data field.

The DCW address constant of LITERAL-N points at the low-order character of the literal. A word mark appears over the high-order character of the literal field.

INDEX REGISTER USAGE

In general, index registers 1-12 are used when executing COBOL object programs, and index registers 13, 14, and 15 are used by the Operating System.

Subprograms – called by entering COMMUNICATION-MODE – do not use all 15 index registers. If a subprogram contains subscripted parameters, as specified in the USING portion of the CALL statement, the subprogram may use index registers 1-7, 9, and 11. If a subprogram does not contain subscripted parameters, the subprogram may use index registers 1-9, 11, and 12. If in the above two cases, the CALL statement that calls the subprogram is not in the DECLARATIVE portion of the Procedure Division, index register 10 may also be used by the subprogram. Index registers that are used by Autocoder subprograms should never be left negative.

Upon returning to the COBOL main object program, index registers 1-15 will be used; the contents of the index registers used by the subprogram are destroyed.

LINKAGE LOADER MEMORY MAP

For the main COBOL object program phase, the four COBOL subprograms must immediately follow the IBCOBOL subprogram on the Linkage Loader memory map. The relocation factors for IBCOBOL and the four COBOL subprograms must be identical. Therefore, when loading user-written subprograms as part of the main COBOL object program phase, a CALLN card must follow the CALL card for COBOL subprogram IIIIIII001. (The I's represent the subprogram identification that is contained in columns 73-79 of the PROGRAM-ID card. This card appears in the IDENTIFICATION DIVISION of the COBOL source program.) In order to force the above order, a CALLN card for one of the COBOL object time subroutines, such as IBCBLDSPLY, should be included as part of the main COBOL object program phase.

EXIT

The EXIT verb is used when it is necessary to provide an end point for a procedure that is to be executed by means of a PERFORM statement, or for procedures specified in the "USE" section. While EXIT is classified as a compiler-directing verb because it supplies the compiler with necessary information and does not produce any coding in the object program, it can also be thought of as a "dummy" program verb.

EXIT must appear in the source program as a one-word paragraph preceded by a paragraph-name. The form of the EXIT verb is:

EXIT.

Further discussion of the EXIT verb is contained in the General Information Manual.

NOTE

The form of the NOTE verb is:

NOTE *any comment.*

See the General Information Manual for additional information concerning this verb.

Ending Verb

STOP

The general form of the STOP verb is:

STOP { *literal* / RUN }

In addition to the details specified in the General Information Manual, the following information pertains to the use of the STOP verb:

1. The statement:

   STOP *literal*

will cause the program to print the literal on the console printer and enter the Wait-Loop routine of the Resident Monitor. (For details, see the publication, *System Monitor.*)

2. The statement:

   STOP RUN

indicates the end of the program and generates the message "STOP RUN" on the console printer, followed by a return to the System Monitor.

Conditional Expressions

In addition to the details contained in the General Information Manual, the following rule applies to conditional expressions.

Within a relational expression the subject, relational operator, and object must all be at the same logical parenthetical level. Therefore, a left parenthesis preceding an object indicates that arithmetic follows.

Example:

VALID	INVALID
IF A = (B + C)	IF A = (B OR C)
IF A = (−B)	IF A = ((B + C) OR D)

Implied subjects and implied relational operators are permissible in conditional expressions. No other abbreviated usage is permitted.

Example:

VALID	INVALID
IF A = B OR C	IF A = B. C OR D

In a conditional expression the logical operator NOT is only permitted at one given parenthetical level.

Example:

VALID	INVALID
NOT (A OR B)	NOT (NOT A OR B)

Added Features of the Procedure Division

The following features, not contained in the General Information Manual, are included in the 1410/7010 COBOL language:

1. The USE Declarative
2. The CORRESPONDING option of the ADD verb
3. The CORRESPONDING option of the SUBTRACT verb

Programming Techniques

When writing COBOL source programs, the techniques described below can be used to produce more efficient machine-language coding or to increase compiling speed.

EFFICIENT OBJECT PROGRAMS

The suggestions given in this section should be followed to obtain the most efficient machine-language coding from the 1410/7010 COBOL compiler. In most cases object program efficiency is significantly improved by using these programming techniques. ADD, COMPUTE, conditional, MOVE, and SUBTRACT statements are subject to the greatest increases in object code efficiency by use of these techniques.

Add: The most efficient ADD statement is:

ADD *data-name-1* TO *data-name-2.*
where the scaling of data-name-1 is identical to that of data-name-2, and the size of data-name-1 is not greater than that of data-name-2.

Compute: A series of simple arithmetic statements usually requires less core storage and is faster than an equivalent COMPUTE expression.

Conditional Statements: Comparisons of unsigned numeric data-names are more efficient than comparisons of signed numeric data-names.

The most efficient conditional statement is:

IF *data-name-1* IS EQUAL TO 'X' GO TO *procedure-name.*
where the size of data-name-1 is one character and 'X' is any one-character alphanumeric literal.

Another efficient conditional statement is:

IF *data-name-1* { IS [NOT] GREATER THAN / IS [NOT] LESS THAN / IS [NOT] EQUAL TO / [NOT] = } *data-name-2*
GO TO *procedure-name.*

where

1. the size of data-name-1 is equal to that of data-name-2;
2. both data-name-1 and data-name-2 are either elementary or redefined;
3. if both data-name-1 and data-name-2 are elementary numeric, then both are unsigned and their scaling is identical.

When an IF and a GO TO statement are not combined, or the condition is part of a PERFORM statement, the most efficient comparison is made where

1. the size of data-name-1 is equal to that of data-name-2;
2. both data-name-1 and data-name-2 are either elementary or redefined;
3. if both data-name-1 and data-name-2 are elementary numeric, then both are unsigned and their scaling is identical.

A series of simple conditional expressions requires less core storage and is faster than an equivalent complex conditional expression. A complex conditional expression is defined in this manual as a conditional expression with one or more of the following: an OR logical operator; parentheses; subscripting; an arithmetic expression.

Move: It is always faster and, in most cases, it requires less core storage to move a level 77 constant than to move a figurative constant.

The most efficient MOVE statement for various types of items and for zero suppression and alphanumeric report editing is:

MOVE *data-name-1* TO *data-name-2*

The following requirements should be fulfilled to obtain the most efficient object coding with this form of the MOVE statement.

For signed, numeric, elementary items, the scaling of data-name-1 should be identical with that of data-name-2.

For unsigned, numeric, elementary items:

1. the scaling of data-name-1 should be identical with that of data-name-2;
2. the size of data-name-1 should be equal to that of data-name-2.

For non-numeric, elementary items, the size of data-name-1 should be equal to or greater than the size of data-name-2.

For group items, the size of data-name-1 should be equal to or greater than that of data-name-2.

For zero suppression:

1. the scaling of data-name-1 should be identical with that of data-name-2;
2. the size of data-name-1 should be equal to that of data-name-2;
3. the PICTURE clause for data-name-1 should contain only the characters 9, V, P, S;
4. The PICTURE clause for data-name-2 should contain only the characters Z, V, P.

For alphanumeric report editing:

1. the scaling of data-name-1 should be identical with that of data-name-2;
2. the PICTURE clause for data-name-1 should contain only the characters 9, V, P, S;
3. the computational size of data-name-1 should be equal to that of data-name-2. "Computational size" refers to the maximum number of numerics

that may be present in the edited item. For example:

EDITING PICTURE	COMPUTATIONAL SIZE
ZZZ.ZZ	5
---.99	4
$$$.99	4
***.99	5

Perform: The most efficient PERFORM option 4 or 5 is:

PERFORM *procedure-name-1* [THRU *procedure-name-2*]

VARYING *data-name-X*

FROM {*literal-1* / *data-name-1*} BY {*literal-2* / *data-name-2*}

UNTIL *condition-1*

where the scaling and length of data-name-X, literal-1 or data-name-1, and literal-2 or data-name-2 are identical.

Subtract: The most efficient SUBTRACT statement is:

SUBTRACT *data-name-1* FROM *data-name-2.*

where the scaling of data-name-1 is identical to that of data-name-2.

Already-Written Programs: For programs that have already been written in COBOL, it is not necessary to examine each COBOL source program statement to determine if the object program efficiency can be improved. The approximate card number and the type of optimization that may be possible are printed as a warning message at compilation time.

In most cases alteration of an operand characteristic or a statement format produces optimum coding.

The list of relocatable core-storage assignments can be obtained by use of the LIST operand option on the MON$$ EXEQ COBOL card. Refer to "1410/7010 COBOL Compiler Requirements" later in this manual.

Miscellaneous: For files that contain multiple records, it may be more economical to define only one form and then transfer the record to an appropriate work area.

Use unsigned numerics whenever possible.

Terminate every source statement with a period.

Use READ INTO and WRITE FROM whenever possible. READ INTO and WRITE FROM should be used only when the receiving area is equal to or larger than the sending area. If the file associated with the READ INTO or WRITE FROM is in Load mode, the word marks will be moved.

For elementary numeric items, the scaling variation should be minimized.

Subscripting and REDEFINES clause usage may be less efficient than other approaches.

Whenever possible, simple statements referencing elementary items should be used rather than complex statements or statements that reference group items.

The ACCEPT and DISPLAY verbs should be used only for low-volume input or output.

EFFICIENT COMPILATION

These suggestions should be followed to increase compilation speed.

1. Unnecessary paragraph-names should be avoided.
2. Certain EXEQ card options (see the section "1410/7010 COBOL Compiler Requirements") cause the compiler to produce additional output. When not essential, these options should not be elected.
3. It is recommended that, wherever possible, the programmer use the PICTURE clause instead of the SIZE, POINT, CLASS, and BLANK clauses of a Record Description entry. The PICTURE clause specifies the characteristics of an elementary item in a more compact form, and can, therefore, be processed more efficiently.

GENERAL CONSIDERATIONS

Some general suggestions and cautions are given below.

1. When desired precision of results of arithmetic expressions exceeds that represented by PICTURE S9(10)V9(10), it is suggested that the appropriate arithmetic verbs be used (i.e., ADD, SUBTRACT, MULTIPLY, and DIVIDE), rather than the COMPUTE verb.
2. The normal contents of the MONITOR-SWITCH, in the Resident Monitor's Communication Region, is a blank. Therefore, it is recommended that the user either:
   a. not assign a blank value to a meaningful condition of this switch; or
   b. let the blank value indicate that the switch has not been set.
3. When a REDEFINES clause is associated with a Load mode input file, the redefined portion of the Load mode record does not carry word marks on tape.
4. Conventions which help to debug source programs are: (1) the placing of all procedure-names on a separate card, and (2) beginning all verbs and conditional statements in column 12 of a card.
5. Tables of constants can be formed by using the OCCURS clause. References to these tables can be made by the use of subscripts. The procedure for constructing tables of constants is described in the *General Information Manual.*

Compatibility Considerations

Certain COBOL verbs and their associated language specifications cannot be defined in compatible terms between the 1410/7010 Systems and other systems. It is suggested that the user avoid the following when writing COBOL programs that are to be compiled on more than one system:

1. ACCEPT
2. UPON option of the DISPLAY verb
3. ENTER
4. USE
5. CALL

For reasons of compatibility, the use of the REDEFINES clause should be limited to one level of redefinition, with the exception that, if the REDEFINES is specified at the 01-level, one additional level of redefinition within the level 01 may be used.

Use of the COBOL Character Set (H2) for literals is suggested, when compatibility with other systems is a consideration.

The following clauses described in the General Information Manual are not implemented by the 1410/7010 COBOL compiler for reasons of compatibility:

1. The JUSTIFIED clause. Standard justification according to CLASS definition will always take place. If nonstandard data manipulation is required, the programmer can use other language specifications for this purpose (e.g., the REDEFINES clause).
2. The Editing clause. Editing functions can only be specified by use of the PICTURE clause.

Qualification of Names

Every name used in a COBOL source program must be unique within the source program, either because no other name has the identical spelling, or because the name exists within a hierarchy of names (so that the name can be made unique by mentioning one or more of the higher levels of the hierarchy). The higher levels are called qualifiers when used in this way, and the process is called qualification.

In addition to the information contained in the General Information Manual covering the qualification of names, the programmer should note the following:

1. Any name that requires qualification, but is not qualified, will refer to the first occurrence of that name in the program. The compiler assigns a relocatable address for each data-name; if there are duplicate data-names, the first address is used. No message is issued in the event of duplicate data-names. Therefore, the programmer must check his program for duplicate data-names and their need for qualification.
2. A name plus all its qualifiers cannot exceed a total of 300 characters. If it does, an error message is produced.

Literals

In addition to the rules for forming literals specified in the General Information Manual, the following rules apply to the 1410/7010 COBOL compiler:

For Forming Numeric Literals: A numeric literal must consist of at least one, and not more than 18 digits. It may also include a sign, preceding the first digit, and/or one decimal point.

For Forming Non-Numeric Literals: Any character in the character set, except the quotation mark, the record mark, and the group mark, can be used in a non-numeric (alphanumeric) literal. Blanks are treated as characters and may be included freely.

A non-numeric literal may occupy more than one line. It is continued by placing a hyphen in the seventh character position of the second card. ("CONT." on the COBOL Program Sheet). The continuation of the non-numeric literal must be preceded by a quotation mark.

Subscripts

In addition to the rules for subscripting, which appear in the General Information Manual, the following applies to the 1410/7010 COBOL compiler:

1. A subscript may not be more than four digits in length.
2. If more than four digits are used, only the low order four digits will be referenced.
3. The compiler does not check the number of digits. It will not issue a diagnostic message if more than four digits are used, if a zero subscript is used, or if the subscript exceeds the limits of the array.
4. The programmer may check his subscripts by means of conditional statements.
5. The length of a subscripted area must not exceed a four-digit number.
6. If a greater area is used, erroneous addresses may be generated.

Character Sets

The IBM Character Set H2 must be used for COBOL source programs. This character set consists of the numerals 0 through 9, the 26 letters of the alphabet, and 12 special characters. The IBM 1410/7010 Character Set may be used only for alphanumeric literals, with the following exceptions: (1) the IBM 1410/7010 character "b" (substitute blank) cannot be used with even-parity tape records; (2) the IBM 1410/7010 character " " (word separator character) cannot be loaded into the IBM 1410 or 7010 with a word mark.

The COBOL (Set H2) special characters are shown below with their equivalents in the IBM 1410/7010 Character Set:

CARD CODE	COBOL (SET H2)	1410/7010 (SET A2)	MEANING
blank			space
11	–	–	minus sign hyphen
12	+	&	plus sign
0-1	/	/	division sign
11-4-8	*	*	multiplication sign check protection symbol
12-4-8	)	□	right parenthesis
0-4-8	(	%	left parenthesis
0-3-8	,	,	comma
11-3-8	$	$	dollar sign
12-3-8	.	.	period decimal point
3-8	=	#	equal sign
4-8	'	@	quotation mark

Figurative Constants

In addition to the details specified in the General Information Manual, the following information pertains to the figurative constants. All figurative constants are treated as belonging only to the ALPHANUMERIC class except where noted.

LOW-VALUE LOW-VALUES	The value of this figurative constant is the space, or blank, the lowest in the collating sequence.
HIGH-VALUE HIGH-VALUES	This figurative constant is defined as the character 9, the highest in the collating sequence.
ZERO ZEROS ZEROES	This figurative constant represents the value 0. It is the only figurative constant that can be treated as belonging to the NUMERIC class or the ALPHANUMERIC class.
SPACE SPACES	This figurative constant represents a blank, or space. It is the only figurative constant that can be treated as belonging to the ALPHABETIC class or the ALPHANUMERIC class.
QUOTE QUOTES	This figurative constant represents the character Note that the use of the word QUOTE to represent the character ' is not equivalent to the use of symbol ' to bound a literal.
ALL "literal"	This figurative constant generates a sequence of characters specified by the single-character non-numeric literal.

TALLY

The word TALLY is the name of a data item whose PICTURE is S99999. It is used primarily to hold information produced by the EXAMINE verb; however, it may be referenced by the programmer in any statement where a signed numeric field is valid.

MONITOR-DATE

In addition to the figurative constants, the IBM 1410/7010 COBOL compiler provides the programmer with the special data-name constant MONITOR-DATE. This data-name constant is the name of a five-character data item (system symbol /DAT/) within the Communication Region of the Resident Monitor. MONITOR-DATE contains the current date established by the System Monitor, and may be used in label-checking routines. The form of the date is yyddd, where: yy is the year (00-99) and ddd is the day of the year (001-366). MONITOR-DATE can be used in the same way as any item described in the Constant Section.

Class Conditions

The General Information Manual specifies that the CLASS of a data item may be NUMERIC, ALPHABETIC or ALPHANUMERIC. In addition, CLASS CONDITION tests can be used for all types of fields to determine, at object time, whether they are wholly numeric or wholly alphabetic in content.

The source statement beginning:

IF FIELD-A IS NUMERIC...

results in a character-by-character check of the value of FIELD-A at object time. If an operational sign is present in the units position, the associated character will be interpreted as being numeric. Thus, −9 is interpreted as "minus 9," not as the letter "R."

The source statement beginning:

IF FIELD-B IS ALPHABETIC...

results in a character-by-character check of the value of FIELD-B at object time. If each character in FIELD-B is alphabetic, the item is considered alphabetic.

Examples: The following table shows how the class of an item is interpreted by the compiler depending on which of the class tests is specified:

CHARACTER	NUMERIC	ALPHABETIC
0-9	YES	NO
SPACE	NO	YES
A-R	YES (if units position)	YES
S-Z	NO	YES
? !	YES (if units position)	NO
Other Special Characters	NO	NO

This section describes the requirements for compilation and execution of COBOL programs under the 1410/7010 Operating System. Knowledge of the contents of the *System Monitor* publication is required for understanding this section.

Requirements for Compilation

In order to process a COBOL source program under the IBM 1410/7010 Operating System, certain control cards are required to direct the operation of the Resident and Transitional Monitors and the Linkage Loader. The required Monitor control cards are:

```
MON$$   JOB
MON$$   MODE
MON$$   EXEQ
MON$$   ASGN
```

The required Linkage Loader control cards are:

```
PHASE
CALLN
CALL
```

These control cards are described in detail in the publication, *System Monitor*. However, certain COBOL options, which are available to the user, are discussed below.

EXEQ Card Operand Options

The user can control the output of the COBOL compiler by placing operands immediately after the comma which follows the third System Monitor option on the EXEQ card. These operands can appear in any order and must be separated by commas, with no intervening blanks. Any of the following operands may be used:

1. LIST—This operand produces a listing of source program names and corresponding object program relocatable storage assignments. If LIST is specified, a check for duplicate procedure-names is made; a warning message is issued if duplicate names exist. If LIST is not specified, no check is made and no warning message issued.

The address assigned to a file-name is that of the low-order character of the File List Origin field of the File Table.

A data-name address is assigned to the low-order character of the data field.

A procedure-name address is assigned to the first character in the paragraph or section.

A subscripted data-name address refers to the base address in the following formula.

$$\text{ACTUAL DATA-NAME ADDRESS} = \text{BASE ADDRESS} + \left(\text{VARIABLE1} \times \text{INCREMENT1}\right) + \left\{\begin{matrix}1D^* \\ \left\{\begin{matrix}2D^* \\ 3D^*\end{matrix}\right.\end{matrix}\right.$$
$$\left(\text{VARIABLE2} \times \text{INCREMENT2}\right) + \left\{\begin{matrix}\\ \left\{\begin{matrix}2D^* \\ 3D^*\end{matrix}\right.\end{matrix}\right.$$
$$\left(\text{VARIABLE3} \times \text{INCREMENT3}\right) \left\{\begin{matrix}\\ \\ 3D^*\end{matrix}\right.$$

*NOTE: Dimensions in array

2. DIAGNOSTIC—This operand suppresses the creation of an object program. (DIAGNOSTIC cannot be requested on the same EXEQ card with TRACE or NOPCH.)

3. TRACE—This operand causes the generation of a self-tracing object program. When each paragraph or section of the main body of the Procedure Division is executed at object time, the paragraph or section-name is printed on the Standard Print Unit.

4. NOPCH—This option should be used only when a Go file is being created. The function of NOPCH is to suppress output on the Standard Punch Unit, thereby providing an object program on the Go file only.

5. NOPRT—This option will suppress the listing of the COBOL source program, warning messages, and error messages on the Standard Print Unit. (NOPRT and LIST cannot be requested on the same EXEQ card.)

In the event of an error in the use of any of these options on the EXEQ card, the compiler will ignore all options, and produce only the normal output (an object program on the Go file and/or the Standard Punch Unit).

Figure 18 shows an EXEQ card for COBOL compilation with the TRACE and LIST options.

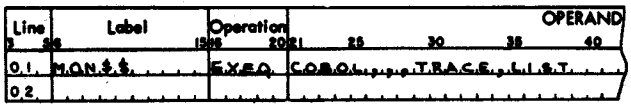


Figure 18. EXEQ Card for COBOL Compilation

Requirements for Execution

The object program produced by the COBOL compiler consists of several subprograms. In accordance with the requirements of the Linkage Loader, each subprogram is headed by a TITLE card.

The Subprogram TITLE Card

The COBOL compiler generates TITLE card information based on the source program.

The format of the TITLE card is:

Column	6	16	21	31	73
	yyddd	TITLE	*PROGRAMnnnxxxx*		*nnnssssss*

where

yyddd is the current date taken from the Resident Monitor's Communication Region.

PROGRAM is the first seven characters of the IDENT field of the PROGRAM-ID card in the source program.

nnn is the subprogram number, assigned serially by the compiler. This number is placed in columns 28-30 and columns 73-75.

xxxxx is the lowest relocatable storage address occupied by the subprogram.

sssss is the sequence-number field of the cards (or card images) in the subprogram. The sequence number of each TITLE card will always be 00001.

The TITLE card is processed in the second phase of the COBOL compiler. The size of COMMON is not known at this time and, therefore, does not appear on the TITLE card. It is not considered by the Linkage Loader and, if core storage limitations are exceeded, the CORE EXCEEDED message is not produced.

IDENT Field of the PROGRAM-ID Card

In order to comply with 1410/7010 Operating System requirements, the following restrictions pertain to completing the IDENT field (columns 73-80):

1. It must always begin with an alphabetic character.
2. It cannot begin with the characters "IB".
3. It cannot contain the slash (/) or any blank characters.

If these requirements are not met, the compiler will replace the erroneous character with "A". For example, if the IDENT field contains IBPSD/b, the TITLE cards will contain IAPSDAAnnn.

Multiple Subprogram COBOL Output

The following list shows the subprogram serial number and function in the normal output of a compilation:

SERIAL NO.	SUBPROGRAM FUNCTION
001	Storage allocation and value declarations for Identification, Environment, and Data Divisions
002	Storage allocation and value declarations for Procedure Division literals
003	Object code for Procedure Division
004	Overlay addresses

COBOL Subroutine Sizes

The sizes of the COBOL object time subroutines are as follows:

SUBROUTINE	NUMBER OF CHARACTERS
IBCOBOL	667
IBCBLCMPAR	1741
IBCBLALTST	91
IBCBLSUBSC	181
IBCBLADOVR	184
IBCBLDSPLY	639
IBCBLACCPT	191
IBCBLEXPON	1160
IBCBLCLEAR	391
IBCBLFLDMP	358
IBCBLRDINT	572
IBCBLDVZER	61

Control Card Requirements

The sequence of the appropriate Monitor and Linkage Loader control cards needed to compile and execute the program with the IDENT "PAYROLL1" using the TRACE option is shown in Figure 19.

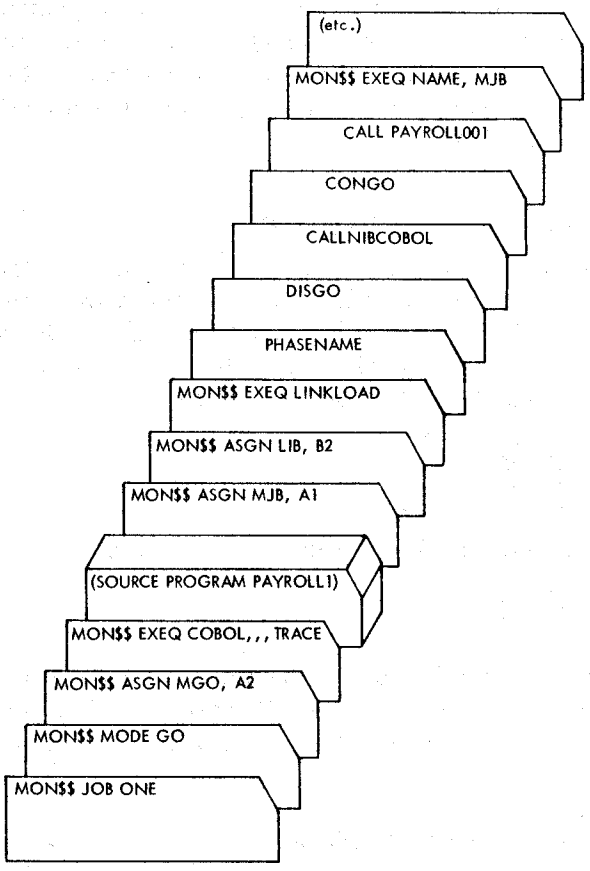


Figure 19. Sample Control Cards for a Compile-and-Go Operation

Linkage Loader control cards DISGO and CONGO eliminate the search of the MGO file for subprogram IBCOBOL. The subprogram IBCOBOL is a required part of every object program. It must be requested with a CALLN card after the PHASE card.

COBOL programs that have been compiled can be added to the System Library file. (For details, see the publication, *System Monitor.*) The CALL requirements for executing these programs from the System Library file are:

```
CALL IBCBLADOVR
CALL IBCBLDSPLY
CALL IBCBLCMPAR
CALL IBCBLSUBSC
CALLN IBCOBOL
CALL PRGNAME001
```

Following the CALLN card for IBCOBOL (Figure 19) is a CALL card for the first of the just-compiled subprograms. The name used in this card consists of the

first seven characters of the IDENT field (PAYROLL) and the serial number, 001. The other three subprograms (PAYROLL002, PAYROLL003, and PAYROLL004) are processed by the Linkage Loader in response to *imbedded* calls that the compiler generates for each set of subprograms.

The Linkage Loader places the relocated program on the Job file, from which it is loaded by the Resident Monitor (EXEQ NAME,MJB). Note that the name used in the EXEQ card for the program must be the same as that used in the PHASE card given the Linkage Loader, but need not be the same as that used in the IDENT field given the compiler.

The sequence of the appropriate Monitor and Linkage Loader control cards necessary to "execute" the program with the IDENT "PAYROLL1", compiled at a *prior* time is shown in Figure 20.

In Figure 20, the object program has been taken from the Standard Punch Unit and is submitted to the Linkage Loader from the Standard Input Unit (rather than from the Go file, as in Figure 19). The basic difference between the two examples (Figures 19 and 20) is that in Figure 20 a CALL card is not used for the subprogram PAYROLL001, because the TITLE card of a subprogram placed in the Standard Input Unit serves the call function.

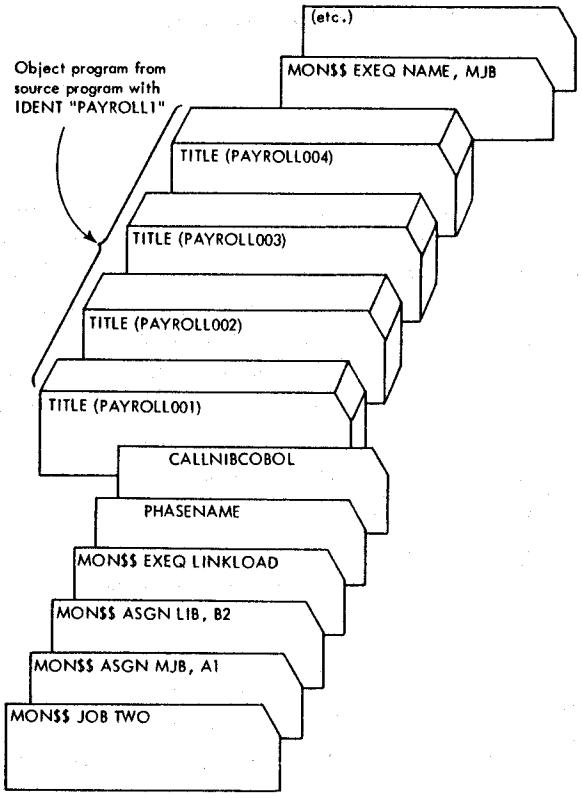


Figure 20. Sample Control Cards for Execution

Display with Carriage Control

It is possible to have carriage control in a COBOL program by making changes to the module IBCBLDSPLY. Depending on how it is done, this may mean sacrificing some of the flexibility of the COBOL program. When the user wishes to display a field, he must first move the field to a buffer area. The Data Division has to set up a one-position field immediately preceding the buffer area, where the user moves in his carriage control character.

Figure 21 shows one way this can be done. In this example, some line count references have been removed, and the user should consider replacing these according to his needs.

```
     MON$$      DATE AUG64
     MON$$      JOB  COBOL CARRIAGE CONTROL
     MON$$      ASGN MW1,B1
     MON$$      ASGN MW2,A2
     MON$$      ASGN MW3,B3
     MON$$      ASGN MW4,A4       HIST FILE IN
     MON$$      ASGN MW5,B5       HIST FILE OUT
     MON$$      ASGN MW6,B6       SG3 OUT, AUTOCODER IN
     MON$$      ASGN MGO,A6
     MON$$      MODE GO
     MON$$      EXEQ SG3
$  $$       HA NUPDATIBCBLDSPLY
$0023A
00238           MLCS 0-1+X1,OUTPUTTANN      USE NEW CC
$0048A,0050A
0048A*          A    OUT,LNCTER      DECREMENT  LINE COUNTER
0048B*          BZ   SETSKIP     TO SKIP TO NEXT PAGE
0048C*          MLCS OUT+6,OUTPUTTANK-1   BLANK CARRIAGE CONTROL
$0056A,0056A
0056AOUTPUTTANNDA   1X133,G
$0057A
0057BOUTPUTTANK     2,2
0057C*OUTPUTTANKDA  1X132,G   OLD OUTPUTTANK REFERENCE
     MON$$      EXEQ AUTOCODER,,MW6,NOFLG
     MON$$      EXEQ COBOL,,,NOPCH,LIST
       IDENTIFICATION DIVISION.
       PROGRAM-ID. CARCONTR.                              CARCONTR
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER. IBM-1410.
       OBJECT-COMPUTER. IBM-1410.
       SPECIAL-NAMES.
           SYSTEM-OUTPUT-PRINTER IS PRT.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       I-O-CONTROL.
       DATA DIVISION.
       01  DUMMY.
       02  CC PICTURE IS X.
       02  BUFFER.
       03  AREA1 PICTURE IS X(50).
       03  AREA2  PICTURE IS X(50).
       03  AREA3 PICTURE IS X(32).
       PROCEDURE DIVISION.
       PARAG-1.
           MOVE 'TEST PICTURE AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
           TO AREA1.
           MOVE 'TEST PICTURE AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA'
           TO AREA2.
           MOVE 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA' TO AREA3.
           MOVE '5' TO CC.     NOTE "2 SPACES AFTER PRINT'.
           DISPLAY BUFFER UPON PRT.
           MOVE 'K' TO CC.      NOTE '2 IMMEDIATE SPACES'.
           DISPLAY BUFFER UPON PRT.
           MOVE '1' TO CC.     NOTE 'IMMEDIATE SKIP TO CHANNEL 1'.
           DISPLAY BUFFER UPON PRT.
           MOVE 'T' TO CC.      NOTE '3 SPACES AFTER PRINT'.
           DISPLAY BUFFER UPON PRT.
           MOVE 'J' TO CC.     NOTE '1 IMMEDIATE SPACE'.
           DISPLAY BUFFER UPON PRT.
       STOP-IT.
           STOP RUN.
     MON$$      ASGN MJB,B3
     MON$$      EXEQ LINKLOAD
                PHASEXYZ
                CALLNIBCBLDSPLY
                DISGO
                CALLNIBCOBOL
                CONGO
                CALL CARCONT001
     MON$$      EXEQ XYZ,MJB
     MON$$      END
```

Figure 21. Carriage Control Subroutine

The purpose of the Autocoder subprogram is to set a Wrong-Length-Record bit on in one or more IORW's of the file, as shown in Figure 22.

The address of the File list origin in the file table is passed from COBOL main program to Autocoder subprogram with the ENTER COMMUNICATION-MODE calling sequence. The File list origin address points to the link field in the IORW. The Autocoder subprogram will determine if there are any alternate areas associated with the file name by testing the link field in the IORW for zero.

The Autocoder subprogram exits back to the COBOL main program after setting a Wrong-Length-Record bit in the last IORW.

```
MON$$      DATE AUG64
MON$$      JOB
MON$$      MODE GO
MON$$      ASGN MGO,A6
MON$$      EXEQ COBOL,SOF,SIU,NOPCH,LIST
  IDENTIFICATION DIVISION.
  PROGRAM-ID. WLRTEST.                                    WLRTEST0
  ENVIRONMENT DIVISION.
  INPUT-OUTPUT SECTION.
  FILE-CONTROL.
      SELECT 90-CHAR-FILE
      ASSIGN TO TAPE-UNIT MR1.
      SELECT 89-CHAR-FILE
      ASSIGN TO TAPE-UNIT MR2
      RESERVE 5 ALTERNATE AREAS.
  DATA DIVISION.
  FILE SECTION.
  FD  90-CHAR-FILE
      RECORDING MODE IS LOAD MODE ODD PARITY
      RECORD CONTAINS 90 CHARACTERS
      LABEL RECORDS ARE OMITTED
      DATA RECORD IS 90-CHAR-RCD.
  01 90-CHAR-RCD.
      02 BSTART PICTURE IS X(30).
      02 MSTART PICTURE IS X(30).
      02 ESTART PICTURE IS X(30).
  FD  89-CHAR-FILE
      RECORDING MODE IS LOAD MODE ODD PARITY
      RECORD CONTAINS 89 CHARACTERS
      LABEL RECORDS ARE OMITTED
      DATA RECORD IS 89-CHAR-RCD.
  01 89-CHAR-RCD.
      02 BBSTART PICTURE IS X(30).
      02 MMSTART PICTURE IS X(29).
      02 EESTART PICTURE IS X(30).
  WORKING-STORAGE SECTION.
  PROCEDURE DIVISION.
  DECLARATIVES.
  TEST-WLR-OPTION SECTION.
      USE AFTER STANDARD ERROR PROCEDURE ON 89-CHAR-FILE.
  1.  DISPLAY 'WLR-89-CHAR-FILE'.
  END DECLARATIVES.
  11. ENTER COMMUNICATION-MODE.
      CALL WLRPGM USING 89-CHAR-FILE.
  12. ENTER COBOL.
  OPEN-90.
      OPEN OUTPUT 90-CHAR-FILE.
      MOVE 'START' TO BSTART.
      MOVE 'MIDDLE' TO MSTART.
      MOVE 'END' TO ESTART.
      WRITE 90-CHAR-RCD.
      CLOSE 90-CHAR-FILE.
      DISPLAY 'UNEQUAL RECORDS'.
  OPEN-89.
      OPEN INPUT 89-CHAR-FILE.
      READ 89-CHAR-FILE
      AT END GO TO STOPITT.
  STOPITT.
      CLOSE 89-CHAR-FILE.
  OPEN-IN-90.
      DISPLAY 'EQUAL RECORDS'.
      OPEN INPUT 90-CHAR-FILE.
      READ 90-CHAR-FILE
      AT END GO TO TNEXT.
  TNEXT.
      CLOSE 90-CHAR-FILE.
      DISPLAY 'EQUAL RECORDS OK'.
  STOPIT.
      STOP 'STOP GIVE DUMP'.
      STOP RUN.
```

```
MON$$      EXEQ  AUTOCODER
           TITLE  WLRPGM
           SBR    X4
START      CW     SETEXIT
           MLCA   4+X4,X5
TEST       MLCB   0+X5,X5
*      TEST LINKFIELD IN IORW FOR ZERO OR ALTERNATE AREAS.
           C      0+X5,0000000
*      ZERO INDICATES LAST IORW
           BE     SETSW
*      MOVE GM TO CHANNEL TEST CHARACTER OR C FIELD
*      OF IORW FOR WRONG LENGTH RECORD CHECK.
                  G
NEXT       MLCS   0M0,19+X5
           NOPWM
SETEXIT    B      NEXTADCON
*      HANDLE NEXT IORW.
           B      TEST
*      PREPARE TO EXIT ON LAST IORW.
SETSW      SW     SETEXIT
           B      NEXT
*      TEST IF LAST FILE HANDLED
NEXTADCON BCE     5+X4,5+X4,N
*      IF NOT REPEAT THE SAME PROCEDURE ON NEXT FILE.
           A      @5@,X4
           B      START
           END
MON$$      ASGN MJB,B3
MON$$      EXEQ LINKLOAD
           DISGO
           CALLNIBCOBOL
           CONGO
           CALL WLRTEST001
MON$$      ASGN MR2,A8
MON$$      ASGN MR1,A8
MON$$      EXEQ IBCOBOL,MJB
MON$$      END
```

Figure 22. Wrong-Length-Record Check Subroutine

The 1410/7010 Operating System multiphase programming feature provides the COBOL programmer with complete segmentation capabilities. For example, if a 41,000-character COBOL program is to be run on a computer that has 40,000 positions of core storage, a low-activity portion of the program can be segmented as a separate phase. The System Monitor publication, in the section on the CHAIN feature, discusses the use of multiphase programming.

For most segmented COBOL programs, the Data Division portion of the program is loaded once. However, segments of the Procedure Division may be loaded many times without disturbing the Data Division. This enables files to be opened in an initialization phase, processed in a series of main phases, and closed in a termination phase.

To use the multiphase programming feature, the user must observe the following general considerations:

1. A phase can consist of COBOL, FORTRAN, or Autocoder routines or any combination of these routines.
2. The number of phases can be up to 999.
3. The number of phases in core storage at one time is dependent upon the number of core storage positions available.

There are three special considerations in implementing COBOL multiphase object programs:

1. The Linkage Loader memory map relocation factors must be equal for all main COBOL subprograms using the same Data Division.
2. COBOL requires a data area (COMMON) at the top of core storage. Care must be taken to avoid overlaying COMMON if Autocoder and/or FORTRAN subroutines (using COMMON) are in core storage with COBOL.
3. In a phase consisting of COBOL object program routines and other language routines, the COBOL routines must contain the entry point.

The relocatable library routine CHAIN loads and executes phases of a program written in one or more of the Operating System languages. The program may be divided into phases by using Linkage Loader control cards.

Illustrations of the Linkage Loader control cards for various combinations of multiphase programs are given in the following examples.

An example of a COBOL, Autocoder, and FORTRAN phase *without* COMMON in the Autocoder or FORTRAN subroutine is illustrated in Figure 23. Note that the Autocoder and FORTRAN subroutines are followed by two COBOL phases.

```
MON$$     EXEQ LINKLOAD
001       PHASENAMEX
          DISGO
          CALL CHAIN
          CALL IBCBLADOVR
          CALL IBCBLDSPLY
          CALL IBCBLCMPAR
          CALL IBCBLSUBSC
          CALLNIBCOBOL
          CONGO
          CALL COBOLP1001
          CALLNAUTOCONAME
          CALL FORTRANAME
002       PHASE
          BASE1COBOLP1002
          CALLNCOBOLP2002
003       PHASE
          BASE1COBOLP2002
          CALLNCOBOLP3002
MON$$     EXEQ NAMEX.MJB
```

Figure 23. COBOL, Autocoder, and FORTRAN Phase *without* COMMON

An example of a COBOL, Autocoder, and FORTRAN phase *with* COMMON in all subroutines is provided in Figure 24.

```
MON$$     EXEQ LINKLOAD
001       PHASENAMEX
          DISGO
          CALL CHAIN
          CALL IBCBLADOVR
          CALL IBCBLDSPLY
          CALL IBCBLCMPAR
          CALL IBCBLSUBSC
          CALLNIBCOBOL
          CONGO
          CALL COBOLP1001
          CALLNIBCBLDVZER
          BASE2(VALUE BELOW COBOL COMMON)
---AUTOCODER PROGRAM IN SIU---
          BASE2(VALUE BELOW AUTOCODER COMMON)
---FORTRAN PROGRAM IN SIU---
          BASE2(VALUE OF TOP OF CORE STORAGE MINUS 2)
002       PHASE
          BASE1COBOLP1002
          CALLNCOBOLP2002
003       PHASE
            .
            .
            .
MON$$     EXEQ NAMEX.MJB
```

Figure 24. COBOL, Autocoder, and FORTRAN Phase *with* COMMON

If a complete overlay of a phase is desired while in a multiphase environment, an Autocoder clear storage subprogram must be loaded as the first subprogram of the new phase. It must clear the new Data Division area before the new Data Division subprogram is loaded.

Figure 25 is an example of the Linkage Loader control cards required for a multi-phase COBOL program with an Autocoder "clear storage" object card routine.

```
     MON$$     EXEQ LINKLOAD
     001       PHASENAMEX
               DISGO
               CALL CHAIN
               CALL IBCBLADOVR
               CALL IBCBLDSPLY
               CALL IBCBLCMPAR
               CALL IBCBLSUBSC
               CALLNIBCOBOL
               CONGO
               CALL COBOLP1001
     002       PHASE
               BASE1COBOLP1001
               TITLEAUTOCDNAME00667                              5
W    W
S00667SYYYYY                                                    NY
                                                                 99999
               BASE1COBOLP1001
               CALLNCOBOLP2001
     MON$$     END
```

Figure 25. Linkage Loader "Clear Storage" Routine

NOTE: In regard to the contents of the Autocoder "clear storage" object card routine, the following designations are to be noted:

00667 – in columns 31-35 of the TITLE card reflects the size of the subprogram IBCOBOL.

5 – in column 72 of the TITLE card is the card type indicator.

YYYYY – in columns 8-12 of the clear storage card equals or is greater than the size of the Phase 002 Data Division, but small enough to fit into core when the relocation factor is added.

N – in column 71 of the clear storage card is the upward relocation indicator.

Y – in column 72 of the clear storage card is the card type indicator.

99999 – in columns 72-76 of the last card is the card type indicator and designation for a subprogram END card.

Figure 26 will produce the equivalent of the Autocoder "clear storage" relocatable object card routine above.

```
MON$$     EXEQ AUTOCODER,,,NONAC
          TITLEAUTOCDNAME
          ORG   00667
          DA    1XYYYYY
          END
```

Figure 26. Autocoder "Clear Storage" Routine

MAIN and SATELLITE Routine

Multiphase programming may be used to retain the Data and Procedure Divisions of the MAIN COBOL program in core storage and to call several SATELLITE Procedure Divisions which will use the Data Division of the MAIN program.

In Figure 27, the MAIN COBOL program calls in the SATELLITE program. SATELLITE returns to the MAIN program and the process is repeated. The run is then terminated in the MAIN program's Procedure Division.

The multiphase program must be compiled and Link loaded as a two-pass system. The first pass determines filler size for "Dummy" entries in the SATELLITE program.

```
     MON$$     DATE SEP64
     MON$$     JOB  COBOL MULTIPHASE
     MON$$     ASGN MW1,B1
     MON$$     ASGN MW2,A2
     MON$$     ASGN MW3,B3
     MON$$     ASGN MGO,A6
     MON$$     MODE GO,TEST
     MON$$     EXEQ COBOL,,,NOPCH,LIST
001010 IDENTIFICATION DIVISION.
       PROGRAM-ID.  MAIN.                                        MAINPGMA
001030 ENVIRONMENT DIVISION.
001040 CONFIGURATION SECTION.
001050 SOURCE-COMPUTER. IBM-1410.
001060 OBJECT-COMPUTER. IBM-1410.
001070 SPECIAL-NAMES. MONITOR-SWITCH ' ' STATUS IS RESET
001080                              'S' STATUS IS SET.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       I-O-CONTROL.
001090 DATA DIVISION.
001100 WORKING-STORAGE SECTION.
001110 77 SWTCH PICTURE IS 9 VALUE IS 1.
001120 77  PHASE-NUMBER PICTURE IS 999 VALUE IS 002.
       01 WORKRECORD.
           02 XXX PICTURE IS X(98).
           02 YYY PICTURE IS X(2).
             88 ZZ VALUE IS '88'.
001140 PROCEDURE DIVISION.
01150 BEGIN. GO TO FIRST-ENTRY SECOND-ENTRY DEPENDING ON SWTCH.
          DISPLAY ' GIVE CORE DUMP PLEASE '.
          STOP 'END OF JOB'.
          STOP RUN.
001170 FIRST-ENTRY. DISPLAY ' MAIN ENTERED. '.
001180 OUT.ENTER COMMUNICATION-MODE CALL CHAIN USING PHASE-NUMBER.
001190 IN.  ENTER COBOL.
002010 SECOND-ENTRY. DISPLAY ' MAIN ENTERED AGAIN '. GO TO OUT.
     MON$$     EXEQ COBOL,,,NOPCH,LIST
001010 IDENTIFICATION DIVISION.
002040 PROGRAM-ID. SATELLITE.                                    SATELLTA
001030 ENVIRONMENT DIVISION.
001040 CONFIGURATION SECTION.
001050 SOURCE-COMPUTER. IBM-1410.
001060 OBJECT-COMPUTER. IBM-1410.
001070 SPECIAL-NAMES. MONITOR-SWITCH ' ' STATUS IS RESET
001080                              'S' STATUS IS SET.
       INPUT-OUTPUT SECTION.
       FILE-CONTROL.
       I-O-CONTROL.
001090 DATA DIVISION.
001100 WORKING-STORAGE SECTION.
001110 77 SWTCH PICTURE IS 9 VALUE IS 1.
001120 77  PHASE-NUMBER PICTURE IS 999 VALUE IS 002.
001130 01  WORKRECORD PICTURE IS X(100).
       01   DUMMY
           02  MAIN-FILLER  PICTURE  X(40).
           88  COMMON-FILLER VALUE 'FILLER'.
           02  REST-OF-MAIN-PROC-DIV  PICTURE X(419).
002120 PROCEDURE DIVISION.
       BEGIN.  DISPLAY ' SATELLITE ENTERED ' SWTCH. ADD 1 TO SWTCH.
       1.   ENTER COMMUNICATION-MODE CALL MAINPGM003.
002150 2.   ENTER COBOL.
     MON$$     ASGN MJB,B3
     MON$$     EXEQ LINKLOAD
               PHASENAME1
               CALLNCHAIN
               CALLNIBCBLADOVR
               CALLNIBCBLDSPLY
               CALLNIBCBLCMPAR
               CALLNIBCBLSUBSC
               CALLNIBCOBOL
               CALL MAINPGM001
               PHASE
               BASE125240
               CALL SATELLT002
     MON$$     EXEQ NAME1,MJB
     MON$$     END
```

Figure 27. MAIN and SATELLITE Routine

One "Dummy" entry is needed to force the SATELLITE program to load above the MAIN program's Procedure Division. So that both programs may reside in core at the same time, SATELLITE must contain a duplicate of the MAIN Data Division and a "Dummy" entry of filler (equal in size to the Procedure Division of MAIN).

A "Dummy" 88 entry is necessary to force the SATELLITE Procedure Division COMMON to be loaded below the MAIN program Procedure Division COMMON, avoiding overlay. The user can determine the size of the COBOL COMMON area by referring to the output listing, which appears on the SPR as a result of the compilation of the program. The number of characters used

is specified at the end of the main program before the memory map.

To calculate the size of the MAIN program's Procedure Division, subtract the address of MAINPGM002 from the address of MAINPGM004.

In Figure 27, the size of "MAIN-COMMON" should be equal to the size of the MAIN program's COMMON area. The size of "REST-OF-MAIN-PROC-DIV" should be equal to the size of the MAIN program's Procedure Division minus the size of COMMON minus one. The BASE1 of the SATELLITE phases should be MAINPGM004.

Appendix A: COBOL Words

The words listed below constitute the complete IBM COBOL vocabulary. Words preceded by an asterisk (*) are not implemented by the 1410/7010 COBOL compiler but should be avoided when assigning names to data, etc., to avoid unnecessary difficulty in converting 1410/7010 COBOL programs to other IBM systems.

Programmers are cautioned that the words recognized by the 1410/7010 COBOL compiler can be used in a COBOL source program only as specified in this publication, or in the General Information Manual.

ACCEPT
ADD
*ADDRESS
*ADDRESSES
AFTER
ALL
ALPHABETIC
ALPHANUMERIC
ALTER
ALTERNATE
AN
AND
APPLY
ARE
AREA
AREAS
ASSIGN
AT
AUTHOR
AUTHORS

*BCD
BEFORE
BEGINNING
BEGINNING-LABEL
*BEGINNING-REEL
*BINARY
BLANK
BLOCK
*BLOCKS
BY

CALL
CARD-PUNCH
CARD-READER
CHARACTER
CHARACTERS
*CHECKPOINT-UNIT
CLASS
CLOSE
COBOL
*COLLATE-MACHINE-SEQUENCE
COMMUNICATION-MODE
COMPUTATIONAL
*COMPUTATIONAL-1
*COMPUTATIONAL-2
COMPUTE
CONFIGURATION
CONSOLE-PRINTER
*CONSOLE-SWITCH
CONSTANT
*CONTAIN
CONTAINS
*CONTROLS
*COPY
CORRESPONDING
*CREATION-DATE
*CREATION-DAY
*CREATION-YEAR

DATA
DATE-COMPILED
DATE-WRITTEN
DECLARATIVES
*DENSITY
DEPENDING
DIGIT
DIGITS
DISPLAY
DIVIDE
DIVISION

*ELECTRONIC-SWITCH
*ELIMINATION
ELSE
END
ENDING
*ENDING-FILE
ENDING-LABEL
ENDING-REEL
END-OF-FILE
END-OF-TAPE
ENTER
ENVIRONMENT
EOF-SIU
EQUAL
ERROR
EVEN
EVERY
EXAMINE
EXIT

FD
FILE
FILES
FILE-CONTROL
FILE-IDENTIFICATION
FILLER
FIRST
FOR
FROM

GIVING
GO
GREATER

*HEADER-LABEL
*HIGH
HIGH-VALUE
HIGH-VALUES
*HYPERTAPE-UNIT
*HYPERTAPE-UNITS

IBM-1410
IBM-7010
IDENTIFICATION
IF
IN
INPUT
INPUT-OUTPUT
INSTALLATION
INTO
I-O-CONTROL
*IOCS
*IOHSK
I-O-SWITCH
IS

LABEL
LEADING
LEFT
LESS
*LIBRARY
LOAD
LOCATION
LOCK
*LONG-LENGTH-RECORD
*LOW
LOW-VALUE
LOW-VALUES

*MEMORY
MODE
MONITOR-DATE
MONITOR-SWITCH
MOVE
*MULTIPLE
MULTIPLY

NEGATIVE
NEXT
NO
*NO-LENGTH-CHECK
*NONE
NON-STANDARD
*NO-OVERLAP
*NO-PRINT-STORAGE
*NO-RELEASE
NOT
*NO-TAPE-MARK
NOTE
NUMERIC

OBJECT-COMPUTER
*OBJECT-PROGRAM
OCCURS
ODD
OF

OFF
OMITTED
ON
OPEN
OPEN-WITHOUT-REWIND
*OPTIONAL-USAGE
OR
OTHERWISE
OUTPUT

PADDING
PARITY
PERFORM
PICTURE
PLACE
PLACES
POINT
POSITIVE
*PREASSEMBLED
PRINTER
*PRIORITY
PROCEDURE
PROCEED
PROGRAM-ID
*PROGRAM-START

QUOTE
QUOTES

READ
RECORD
RECORDING
*RECORD-MARK
RECORDS
REDEFINES
REEL
*REELS
*REEL-SEQUENCE-NUMBER
*REFERENCE
REMARKS
RENAMING
REPLACING
RERUN
RESERVE
RETENTION-PERIOD
REWIND
RIGHT
ROUNDED
RUN

SECTION
SECURITY
SELECT
SENTENCE
*SHORT-ALPHA-WORD
*SHORT-LENGTH-RECORD
SIGNED
SIZE
SOURCE-COMPUTER
SPACE
SPACES
SPECIAL-NAMES
STANDARD
STATUS
STOP
SUBTRACT
*SUPERVISOR
SYNCHRONIZED
*SYSTEM-INPUT-UNIT
SYSTEM-OUTPUT-PRINTER
SYSTEM-OUTPUT-PUNCH

TALLY
TALLYING
TAPE-UNIT
*TAPE-UNITS
THAN
THEN
THROUGH
THRU
TIME
TIMES
TO
*TRAILER-LABEL
*TYPEWRITER
UNEQUAL
*UNIT-RECORD-I-O-RECORD
UNTIL
UPON
USAGE
USE
USING

VALUE
VARYING

WHEN
WITH
*WITH-LABELS
*WITHOUT-LABELS
*WORDS
WORKING-STORAGE
WRITE

ZERO
ZEROES
ZEROS

Appendix B: Organization of Source Program

Some items which may appear in a source program are required, while others are optional. Whether an item is required or optional may be determined by reading the discussion of each individual COBOL word in this publication. The order of appearance of the divisions is mandatory and all divisions must be present. Certain sections within the divisions must also appear as specified, while others have no rigid rules. The items which may appear in a source program are the following:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. program-name.
AUTHOR. author-name.
INSTALLATION. ...
DATE-WRITTEN. ...
DATE-COMPILED. ...
SECURITY. ...
REMARKS. ...

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. ...
OBJECT-COMPUTER. ...
SPECIAL-NAMES. ...
INPUT-OUTPUT SECTION.
FILE-CONTROL. SELECT ...
I-O-CONTROL. APPLY ...

DATA DIVISION.
FILE SECTION.
FD file-name-1 ...
   01 data-name-1 ...
      02 data-name ...
         03 data-name ...
         88 condition-name ...
               .
               .
               .
      02 data-name ...
            .
            .
            .
   01 data-name ...
         .
         .
         .
FD file-name-2 ...
      .
      .
      .
FD file-name-n ...
      .
      .
      .

WORKING-STORAGE SECTION.
77 data-name ...
   88 condition-name ...

         .
         .
         .
77 data-name ...
      .
      .
      .
```

```
01 data-name ...
   02 data-name ...
          .
          .
          .
01 data-name ...
   02 data-name ...
      03 data-name ...
      88 condition-name ...
             .
             .
             .
   02 data-name ...
          .
          .
          .
01 data-name ...
      .
      .
      .
CONSTANT SECTION.
77 data-name...
      .
      .
      .
77 data-name ...
01 data-name ...
   02 data-name ...
          .
          .
          .
01 data-name ...
   02 data-name ...
      03 data-name ...
             .
             .
             .
   02 data-name ...
          .
          .
          .
01 data-name ...
      .
      .
      .
PROCEDURE DIVISION.
DECLARATIVES.
section-name SECTION. USE ...
paragraph-name. ...
END DECLARATIVES.
paragraph-name. ...
      .
      .
      .
section-name-1 SECTION.
paragraph-name-1. ...
      .
      .
      .
paragraph-name-2. ...
      .
      .
      .
paragraph-name-n. ...
      .
      .
      .
section-name-2 SECTION.
      .
      .
      .
section-name-n SECTION.
paragraph-name-1. ...
      .
      .
      .
paragraph-name-2. ...
      .
      .
      .
paragraph-name-n. ...
      .
      .
      .
```

Appendix C: Object Time Error Analysis and Messages

The following object time conditions will cause immediate job termination. The messages will appear on the Standard Print Unit and control will be transferred to the Resident Monitor's Unusual End of Program. (For details see the *System Monitor* publication.)

In the following messages, if the program being run was compiled in the TRACE mode, the name of the paragraph in which this error occurred appears as the last paragraph-name on the Standard Print Unit.

If the program being run was *not* compiled in the TRACE mode, nnnnn is the relocated address of the statement in error.

INVALID EXPONENTIATION nnnnn

An attempt to raise zero to the zero power has been detected.

ZERO DIVISOR nnnnn

An attempt to divide by zero has been detected.

UNALTERED STATEMENT nnnnn

A GO TO statement which required ALTERing was executed prior to being ALTERed.

Appendix D: Diagnostic Messages

This appendix includes all the diagnostic messages produced by the 1410/7010 COBOL compiler, and their meanings. The messages are listed by division, with a "general" section for messages which can occur in more than one division.

Normally, when a diagnostic message appears on the source program listing, an incomplete object program will be produced. The compiler will continue to examine the entire source program for further errors but terminates object program output. However, some messages are merely warnings to the programmer, and do not necessarily affect the compilation of the source program. A "W" preceding a message in this appendix indicates a warning-type message. Also indicated (for warning-type messages) are any assumptions the compiler may make about the intent of the statement in question.

Source Program Listing

The following information is included to assist the programmer to better understand the source program listing:

1. An "S" appearing to the left of a statement, or group of statements, indicates that the programmer-supplied sequence numbers are out of sequence. (This is a warning and does not affect compilation.)

2. The four-digit number appearing on the extreme left of the source program Procedure Division listing is a card reference number assigned by the compiler.

3. Source statements and diagnostic messages for the Identification, Environment, and Data Divisions appear interspersed in the source program listing.

4. Diagnostic messages for the Procedure Division appear in one section of the source program listing, with a card number to the left of the diagnostic message. The diagnostic message usually refers to the statement or procedure-name on the indicated card. However, because of the object code optimizers lookahead subprogram, the message may refer to a statement or procedure-name on an earlier card.

The lookahead subprogram scans the COBOL source program twenty data-name or key-word elements ahead of the object code output routine. Therefore, under most circumstances, the diagnostic message will refer to the current card or one of the three preceding cards.

5. A message will appear indicating the storage allocated to the main program. This allocation does not include any optional subprograms called by the user or by the compiler.

6. One of the following messages appears on the Standard Print Unit if compilation of the source program has been prevented:

a. SHORT LENGTH WORK FILE (I/O operation)

Appears if one of the work files (MW1, MW2, or MW3) assigned for the COBOL compiler is not of sufficient length. Compilation is terminated and control passes to the Resident Monitor's Special-End-of-Program routine. (See the publication, *System Monitor*, for details.)

b. ******OBJECT PROGRAM INCOMPLETE******

Appears if there are errors in the source program which cannot be corrected by the COBOL compiler. It is the terminating message of a COBOL compilation in which a source error prevented the generation of a complete object program, unless DIAGNOSTIC has been specified on the EXEQ card. (See 7, below.) If this is a compile-and-go operation, the Go file is cancelled.

c. UNCORRECTABLE I/O ERROR IN (phase name)

Appears if there is some uncorrectable input or output error. Compilation is terminated and control passes to the Resident Monitor's Special-End-of-Program routine. (See the publication, *System Monitor*, for details.)

In conjunction with messages a, b, and c, the following messages appear on the console printer:

10980 SHORT LENGTH WORK FILE (I/O operation)
10990 OUTPUT INCOMPLETE—SOURCE ERROR
10999 OUTPUT INCOMPLETE—I/O ERROR
(phase name)

d. ******SOURCE PROGRAM INCOMPLETE******

Appears if the four COBOL Divisions are not found in the source program. Compilation is terminated and control passes to the Resident Monitor's Special-End-of-Program routine. (See the publication, *System Monitor*, for details.) If this is a compile-and-go operation, the Go file is cancelled.

e. NUMBER OF ENTRIES WITHIN GROUP EXCEEDS TABLE SIZE IN (phase name)

Appears if the table limit is exceeded. The COBOL compiler requires a minimum of 28,500 positions of core storage, allowing for an 11,500-position Resident Monitor on a system that has a 40,000 character core-storage unit. Within a single 01 record, up to 100 data-name entries may usually be processed. However, in some cases the limit may be higher or lower depending upon the characteristics of the data names and the actual Monitor size.

If this message appears while processing the Data Division, the user should either reduce the number of entries in the 01 record or reduce the size of the Monitor. Reducing the size of the Monitor will increase the table size. If this message appears while processing the Procedure Division, the user should reduce the number of entries within the FD.

The above message should not appear for systems in which more core storage is available for the table.

7. The following message appears on the Standard Print Unit, and is the terminating message for a COBOL "DIAGNOSTIC" run.

******END OF DIAGNOSTIC RUN******

General

The messages appearing below can occur in more than one division.

W DUPLICATE CLAUSE KEYWORD
A clause keyword has appeared more than once in an entry.

W INVALID CONTINUATION CARD
Blank continuation card. It is ignored.

W INVALID LITERAL SYNTAX
Record mark or group mark found in a VALUE clause or a non-numeric literal format error.

W KEYWORD DIVISION MISSING
Presence of the word DIVISION is assumed.

W REFERENCE FORMAT ERROR
One of the format rules has been broken but is ignored by the compiler. (See the General Information Manual.)

W SYNTAX CHECK DISCONTINUED WITH "X"
The syntactical form of the input statements does not conform to COBOL syntax. "X" is the word initiating the erroneous source data.

W SYNTAX CHECK RESUMED WITH "X"
A valid syntactical form is recognized after the occurrence of the above message. "X" is the word with which checking is resumed.

Identification Division

W INVALID IDENTIFICATION ASSIGN IDENTIFICATION "X"
Columns 73-80 invalid. Value of "X" is assigned by the compiler.

W SEARCH FOR IDENTIFICATION
The keyword IDENTIFICATION has not been discovered in its proper position.

W SEARCH FOR PROGRAM-ID
The keyword PROGRAM-ID has not been discovered in its proper position.

Environment Division

W ASSIGN CLAUSE MISSING
Self-explanatory.

W CONFIGURATION SECTION MISSING
Self-explanatory.

W CONFIGURATION SECTION OUT-OF-ORDER
Self-explanatory.

W DUAL OBJECT-COMPUTER SPECIFICATION
IBM 1410 and IBM 7010 specified.

W DUAL SOURCE-COMPUTER SPECIFICATION
IBM 1410 and IBM 7010 specified.

W FILE-CONTROL PARAGRAPH MISSING
Self-explanatory.

W INPUT-OUTPUT SECTION MISSING
Self-explanatory.

W INVALID APPLY CLAUSE SYNTAX
The clause is ignored.

W INVALID APPLY LITERAL
More than one padding character, or invalid padding character.

W INVALID ASSIGN CLAUSE SYNTAX
The clause is ignored.

W INVALID DEVICE-NAMES CLAUSE SYNTAX
The clause is ignored.

W INVALID LABEL RECORD SIZE
Beginning-Label or Ending-Label record is greater than 120 characters.

W INVALID RENAMING CLAUSE SYNTAX
The clause is ignored.

W INVALID RERUN CLAUSE SYNTAX
The clause is ignored.

W INVALID RESERVE CLAUSE SYNTAX
The compiler assumes there are NO ALTERNATE AREAS.

W INVALID SELECT SYNTAX
SELECT not followed by ASSIGN, RESERVE, or RENAMING.

W INVALID SPECIAL-NAMES PARAGRAPH SYNTAX
The paragraph is ignored.

W INVALID SWITCH-NAMES CLAUSE SYNTAX
The clause is ignored.

W KEYWORD SECTION MISSING
Self-explanatory.

W MISSING PERIOD
Self-explanatory.

W NO I-O-CONTROL PARAGRAPH
Self-explanatory.

W NO SECTION HEADING
Self-explanatory.

W OBJECT-COMPUTER PARAGRAPH MISSING
Self-explanatory.

W PARAGRAPH INVALID IN THIS SECTION
Processing will take place as if SECTION were correct.

W PARAGRAPH OR STATEMENT CONSTRUCTION ERROR
Self-explanatory.

W PARAGRAPH OUT-OF-ORDER
Self-explanatory.

W SOURCE-COMPUTER PARAGRAPH MISSING
Self-explanatory.

W UNDEFINED RENAMING FILE-NAME
File used in APPLY clause not defined.

W UNDEFINED RENAMING, FILE-NAME
File used in RENAMING clause not defined.

W UNDEFINED RERUN FILE-NAME
File used in RERUN clause not defined.

Data Division

W 77-LEVEL OUT-OF-ORDER
Self-explanatory.

W 88-LEVEL INVALID AT GROUP LEVEL
Self-explanatory.

W 88-LEVEL INVALID IN THIS SECTION
An 88-level appears in the CONSTANT SECTION. This entry is ignored.

W CLAUSE MISSING IN THIS FD
LABEL RECORDS clause is missing and is assumed to be omitted; or DATA RECORDS clause is missing and is ignored.

W FD ENTRY RECORD MISSING
An FD entry has no associated Record Description items. File is ignored.

W FD OUT-OF-ORDER
FD has been detected in other than File Section. Compiler will handle this condition.

W FILE SECTION OUT-OF-ORDER
Will be processed as if in proper order.

W INCOMPATIBLE BLOCK RECORD CLAUSE
Combination of BLOCK CONTAINS and RECORD CONTAINS clause does not agree with one of the five allowable formats.

W INCOMPATIBLE BLOCK RECORD SIZE
The record size is too large. Record size will be used.

W INCOMPATIBLE CLASS PICTURE CLAUSE
Classes as specified by the CLASS and PICTURE clauses in a given item do not agree. CLASS clause is ignored.

W INCOMPATIBLE LITERAL
CLASS or PICTURE does not agree with VALUE literal. The compiler ignores this condition and allocates storage for the literal.

W INCOMPATIBLE PICTURE POINT CLAUSE
The assumed decimal point in the POINT clause does not agree with the PICTURE. POINT clause is ignored.

W INCOMPATIBLE POINT CLASS CLAUSE
The POINT clause is not associated with a numeric item. POINT clause is ignored.

W INCOMPATIBLE RECORD SIZE
The record size as derived from the Record Description does not agree with the size as stated in the RECORD CONTAINS clause. The computed record size will be used.

W INCOMPATIBLE REDEFINES ENTRY
If the rules outlined below are not applicable, the message may be ignored.
1. The redefinition cannot exist at a 01 level number in the File Section.
2. A data-name which is subscriptable cannot be redefined.
3. The size associated with the redefinition is greater than the size of the original area. The size of the original is used.

W INCOMPATIBLE SIGNED PICTURE CLASS CLAUSE
The existence of a sign, specified by the SIGNED clause, does not agree with the PICTURE, which is non-numeric. The SIGNED clause is ignored.

W INCOMPATIBLE SIZE CLAUSE AT GROUP LEVEL
SIZE as specified at group level does not agree with the size as calculated from the contained elementary items. Group SIZE is made to conform.

W INCOMPATIBLE SIZE PICTURE CLAUSE
Size as specified in a SIZE clause does not agree with the size given by the PICTURE clause. SIZE clause is ignored.

W INCOMPATIBLE WITH HIGHER LEVEL CLASS
Class specified for this item does not agree with class specified for the group. Group CLASS is ignored.

W INVALID 88-LEVEL
88-level occurs without a preceding condition variable (valid level-number). It is ignored.

W INVALID BLOCK CLAUSE SYNTAX
The compiler will infer block size from record size.

W INVALID CLASS SYNTAX
CLASS clause is ignored.

W INVALID DATA RECORD CLAUSE SYNTAX
The clause is ignored.

W INVALID DEPENDING ON ENTRY
The DEPENDING ON data name within a given RECORD CONTAINS clause either does not occur in a subsequent file record, or does not have consistent specifications in a multi-record file.

W INVALID EDITING CLAUSE SYNTAX
Invalid BLANK WHEN ZERO clause.

W INVALID LABEL RECORD CLAUSE SYNTAX
Compiler assumes OMITTED.

W INVALID LEVEL-NUMBER
The level-number of the first item following an FD is not 01. This item is assumed to be a 01-level.

W INVALID LEVEL-NUMBER SYNTAX
Invalid level-number sequence. Will be treated as if valid; therefore, hierarchical relationships may be affected.

W INVALID LITERAL
File Identification value is improper. If more than 10 characters the value is truncated.

W INVALID LITERAL IN THIS CONTINUATION CARD
Continuation indicator, but first non-blank character, not the quote sign. Continuation ignored-literal is terminated by end of first card.

W INVALID LITERAL SYNTAX
Literal in VALUE clause is invalid. VALUE clause is ignored.

W INVALID OCCURS CLAUSE
OCCURS clause generates a fourth or higher dimension array. The clause is ignored.

W INVALID OCCURS CLAUSE SYNTAX
The clause is ignored.

W INVALID PERIOD
Self-explanatory.

W INVALID PICTURE SYNTAX
The clause is ignored.

W INVALID POINT CLAUSE SYNTAX
The clause is ignored.

W INVALID PUNCTUATION OR SPECIAL CHARACTER
This is ignored.

W INVALID RECORD SYNTAX
Syntactical error in RECORD CONTAINS clause. The clause is ignored.

W INVALID RECORDING MODE CLAUSE SYNTAX
Compiler assumes Move mode and even parity.

W INVALID REDEFINES CLAUSE SYNTAX
The redefined data name is undefined or the entries redefining an area do not immediately follow the original definition of the area, or the redefined data-name level-number does not agree with the current-name level-number.

W INVALID SIZE CLAUSE SYNTAX
The clause is ignored.

W INVALID SYNCHRONIZED CLAUSE SYNTAX
The clause is ignored.

W INVALID U/R SPECIFICATION
Recording mode specified is invalid for unit record. Move mode and even parity is assumed.

W INVALID USAGE CLAUSE SYNTAX
The clause is ignored.

W INVALID VALUE CLAUSE
VALUE and REDEFINES clauses are in same item. VALUE and OCCURS clauses are in same item. VALUE in item subordinate to grouped REDEFINES item. VALUE in item subordinate to grouped OCCURS item. VALUE within a File Section Record Description; or VALUE with report item. The VALUE is ignored.

W KEYWORD SECTION MISSING
The word SECTION does not appear. The compiler assumes that it is present.

W LITERAL EXCEEDS MAXIMUM CHARACTER SIZE 120
Literal will be truncated.

W LITERAL TRUNCATION
VALUE exceeds SIZE. This message will also appear whenever a VALUE is given to a field whose PICTURE includes PICTURE symbol "P" on the left.

W NO CONTINUATION CARD INVALID LITERAL
No terminal quote sign on current card, or no continuation indicator on next one. Literal assumed terminated at end of first card.

W NO ENTRY CLASS
No CLASS or PICTURE for an elementary item. Low order character(s) of the literal will not fit in the field as specified.

W NO LITERAL WITH 88-LEVEL
88 is assigned a value of blanks.

W NO SIZE IN THIS ENTRY
Self-explanatory.

NUMBER OF ENTRIES WITHIN GROUP EXCEEDS TABLE SIZE–BREAK UP GROUP USING REDEFINES OPTION
Self-explanatory.

W OCCURS CLAUSE INVALID IN THIS ENTRY
OCCURS clause associated with a 01 or 77-level item. The clause is ignored.

W PICTURE CLAUSE INVALID AT GROUP LEVEL
PICTURE clause is describing a group item rather than an elementary item. The clause is ignored.

W POINT CLAUSE INVALID AT GROUP LEVEL
POINT clause is used to describe group rather than elementary item. The clause is ignored.

W PUNCTUATION INVALID IN THIS ENTRY
One of the punctuation rules has been broken. (See the General Information Manual.) Punctuation is ignored.

W RECORD CLAUSE MISSING
RECORD CONTAINS clause is missing.

RECORD OUT-OF-ORDER
A Record Description entry within the File Section has no associated FD. Item is processed as WORKING-STORAGE.

W REDEFINES CLAUSE OUT-OF-ORDER
REDEFINES clause is not the first clause in an item. The clause is accepted.

W REDUNDANT 88-LEVEL CLAUSE
A clause other than VALUE is associated with an 88-level item. This is ignored.

W SIGNED CLAUSE INVALID AT GROUP LEVEL
SIGNED clause is used to describe group item rather than elementary item. This is ignored.

W UNDEFINED DATA-RECORD
01 Record not defined in DATA RECORD clause.

W UNDEFINED ENTRY
Undefined name in REDEFINES clause or RECORD CONTAINS DEPENDING ON clause.

W UNDEFINED FILE
FD entry has no associated SELECT clause, or invalid SELECT clause.

W VALUE CLAUSE INVALID AT GROUP LEVEL
The clause is ignored.

W WORD EXCEEDS MAXIMUM CHARACTER SIZE 30
The word is truncated.

W WORKING STORAGE SECTION OUT-OF-ORDER
This is processed as if in proper order.

Procedure Division

W ARITHMETIC OPTIMIZATION POSSIBLE
Refer to "Programming Techniques."

(name) IS AN INVALID QUALIFIER IN (name)
Invalid qualifier is identified by the paragraph in which it is used.

(name) IS AN UNDEFINED NAME IN (name)
Undefined procedure-name is identified by the paragraph in which it is used.

(name) NOT A CONDITION-NAME
Self-explanatory.

(name) OVERSIZE PARAGRAPH
Paragraph should be broken down into more than one paragraph.

W CONDITIONAL CLASS CONTRADICTION
Data items of unlike class are being compared, or a non-numeric data item is being tested for a sign, or a sign test on an unsigned numeric data item.

W CONDITIONAL OPTIMIZATION POSSIBLE
Refer to "Programming Techniques."

W CORRESPONDING OPERATOR INVALID
REPLACED WITH "X" OPERATOR
In MOVE, ADD, or SUBTRACT CORRESPONDING, TO or FROM was missing. "X" will be either "TO" or "FROM."

CORRESPONDING STATEMENT FORMAT ERROR
Self-explanatory.

W CORRESPONDING VERB IGNORED
CORRESPONDING used with other than MOVE, ADD, or SUBTRACT.

EXAMINE OPERAND ERROR
Attempt to EXAMINE a constant or a literal.

GO TO STATEMENT MISSING DEPENDING
Self-explanatory.

INCORRECT CONDITIONAL EXPRESSION
Self-explanatory.

W INCORRECT CONTINUATION
Continuation card error. Text starts prior to column

12 of continuation card. Unnecessary continuation indicator detected. This condition is ignored.

W INCORRECT END DECLARATIVES
The compiler will correct this error.

INCORRECT LITERAL
Invalid record mark or group mark.

INCORRECT LITERAL CONTINUATION
Non-numeric literal continuation error.

W INCORRECT PUNCTUATION
Incorrect punctuation will be ignored.

W INPUT OPTIMIZATION POSSIBLE
Refer to "Programming Techniques."

W INVALID ALTER STATEMENT
Something other than a Paragraph/Section-name follows ALTER; TO PROCEED TO is not specified properly; Paragraph/Section-name does not follow TO PROCEED TO; invalid format for compound ALTER statements or more than one level of qualification has been given for Paragraph/Section-name.

INVALID CALL STATEMENT
Self-explanatory.

INVALID CHARACTER
1410/7010 special character meaningless to COBOL will be ignored.

INVALID COMPUTE OPERAND
Self-explanatory.

INVALID COMPUTE OPERATOR
Self-explanatory.

INVALID CONDITIONAL OPERAND
Data-name is used incorrectly.

INVALID CONDITIONAL OPERATOR
Self-explanatory.

INVALID CORRESPONDING
CORRESPONDING option is used incorrectly.

INVALID DATA-NAME IN ACCEPT STATEMENT
Self-explanatory.

INVALID DECLARATIVES
Section-name does not follow DECLARATIVES. The compiler will skip to the next procedure-name or END DECLARATIVES.

INVALID DISPLAY DEVICE
Self-explanatory.

INVALID ENTER STATEMENT
Previous ENTER statement missing or syntax error.

INVALID EXAMINE STATEMENT
Self-explanatory.

INVALID EXIT
Keyword EXIT appeared in other than a one-word paragraph.

INVALID OPERAND
The cause of the message can be: invalid operands, invalid qualification, more than three levels of subscripting, data-name class omitted for procedure statements, or an invalid operand for an ADD verb referencing a group item.

INVALID OPERAND AFTER GIVING CLAUSE
Multiple receiving fields invalid.

INVALID OPERAND USAGE IN CORRESPONDING
Multiple receiving field specified in ADD or SUBTRACT CORRESPONDING, or invalid data-name, such as literal or elementary item used, or a level 77 used.

INVALID PARENTHESIS
Self-explanatory.

INVALID PERFORM STATEMENT
Self-explanatory.

INVALID STATEMENT
Missing ENTER COBOL.

INVALID USE STATEMENT
Self-explanatory.

INVALID WORD AFTER OPEN VERB
Self-explanatory.

IS UNDEFINED
Undefined name. The name will appear on the preceding line.

W LITERAL EXCEEDS 120 CHARACTERS
Literal will be truncated.

MISSING OPERANDS IN CORRESPONDING
If LIST option is specified, this informative message appears and is followed by a list of the data-names that satisfy the requirements for a match in the CORRESPONDING option.

MISSING AT END IN READ STATEMENT
Self-explanatory.

MISSING BY AFTER VARYING IN PERFORM STATEMENT
Self-explanatory.

MISSING DISPLAY OPERAND ONE
Self-explanatory.

MISSING ERROR AFTER SIZE
Self-explanatory.

MISSING FIRST MOVE OPERAND
Self-explanatory.

MISSING FROM AFTER VARYING IN PERFORM STATEMENT
Self-explanatory.

MISSING IF TO MATCH THIS NEXT SENTENCE CLAUSE
Self-explanatory.

MISSING LEFT PARENTHESIS IN CONDITIONAL
Self-explanatory.

MISSING LITERAL IN EXAMINE STATEMENT
Self-explanatory.

MISSING LITERAL TWO AFTER EXAMINE
Self-explanatory.

MISSING OPERAND ONE IN THIS STATEMENT
Self-explanatory.

W MISSING PERIOD BEFORE P/S NAME
Statement not properly terminated before new Paragraph/Section-name.

MISSING PERIOD OR SECTION AFTER PROCEDURE-NAME
Self-explanatory.

MISSING PROCEDURE IN USE STATEMENT
Self-explanatory.

MISSING PROCEDURE-NAME AFTER GO TO
Self-explanatory.

MISSING PROCEDURE-NAME IN PERFORM STATEMENT
Self-explanatory.

MISSING RECEIVING OPERAND
Self-explanatory.

MISSING REPLACING OR BY IN EXAMINE STATEMENT
Self-explanatory.

MISSING REWIND AFTER NO
Self-explanatory.

MISSING RIGHT PARENTHESIS IN CONDITIONAL
Self-explanatory.

MISSING RUN OR LITERAL AFTER STOP
Self-explanatory.

MISSING SECOND MOVE OPERAND
Self-explanatory.

MISSING SENTENCE AFTER NEXT
Self-explanatory.

MISSING STATEMENT 1 TO MATCH THIS OTHERWISE OR ELSE
The word ELSE or OTHERWISE is used without an associated IF statement.

MISSING TALLYING OR REPLACING IN EXAMINE STATEMENT
Self-explanatory.

MISSING TIMES IN PERFORM STATEMENT
Self-explanatory.

MISSING TO AFTER GO
Self-explanatory.

MISSING TO AFTER MOVE
Either the TO after the MOVE verb was missing, or a subscript error appeared in the first operand after MOVE.

MISSING UNTIL AFTER VARYING IN PERFORM STATEMENT
Self-explanatory.

MISSING VALID CONDITIONAL OPERAND
Self-explanatory.

MISSING VALID EXPONENTIATE OPERAND
Self-explanatory.

MISSING VALID FILE-NAME AFTER CLOSE VERB
Self-explanatory.

MISSING VALID FILE-NAME AFTER OPEN INPUT
Self-explanatory.

MISSING VALID FILE-NAME AFTER OPEN OUTPUT
Self-explanatory.

MISSING VALID FILE-NAME IN READ STATEMENT
Self-explanatory.

MISSING VALID GO TO DEPENDING OPERAND
Data-name is missing or is not an integer.

MISSING VALID OPERAND AFTER BY OR INTO
Self-explanatory.

MISSING VALID OPERAND AFTER EXAMINE
Self-explanatory.

MISSING VALID OPERAND AFTER GIVING
Self-explanatory.

MISSING VALID OPERAND AFTER VARYING IN PERFORM
Self-explanatory.

MISSING VALID READ AREA-NAME
Self-explanatory.

MISSING VALID WRITE AREA-NAME
Self-explanatory.

MISSING VALID WRITE RECORD
Self-explanatory.

W MOVE CLASS CONTRADICTION
Self-explanatory.

MOVE OPERAND ERROR
The receiving field designated is a literal, constant, etc.

W MOVE OPTIMIZATION POSSIBLE
Refer to "Programming Techniques."

MOVE SUBSCRIPT FROM OPERAND
More than two subscripted data-names have appeared in the USING option of the CALL verb.

NO MATCH FOR CORRESPONDING
No match found for MOVE (one item elementary) or Arithmetic (both items elementary numeric). Improper qualifications exist for matching data items. Matching data items are in secondary redefined area, or qualified by same.

W OUTPUT OPTIMIZATION POSSIBLE
Refer to "Programming Techniques."

PARAGRAPH/SECTION INCOMPLETE IN (name)
This message occurs if a source error has prevented processing of part of a statement or paragraph; or a statement implies the existence of a clause or statement that is not present.

W POSSIBLE TRUNCATION
Sending or FROM data-name larger than receiving data-name, or storing of arithmetic results where digits might be lost. (This message may occur where the ROUNDING option is used, and should be ignored.)

P/S NAME FORMAT ERROR
Procedure-name not followed by SECTION or period.

QUALIFIED NAME EXCEEDS STORAGE ALLOCATION
Total number of characters has exceeded 300.

S$$ IS AN UNDEFINED NAME
The DECLARATIVE SECTION is used and the END DECLARATIVES card is missing, or the key words misspelled. S$$ is assigned as the name for the card and is detected as an undefined name.

SUBSCRIPT ERROR
Subscripting used with a data-name not associated with an OCCURS clause, or the number of subscripts used does not agree with the associated data description.

USE VERB MISSING
In DECLARATIVES, first word after section-name SECTION must be USE. Compiler will skip to the next procedure-name or END DECLARATIVES.

W WORD EXCEEDS 30 CHARACTERS
Word is truncated.

IBM

COBOL PROGRAM SHEET

Form No. X28-1464 Printed in U.S.A.

PAGE 1 3 | PROGRAM SAMPLE PROBLEM 1410/7010 COBOL | SYSTEM 1410 | SHEET 1 OF 7
001 | PROGRAMMER | DATE | IDENT 73 PAYROLL1 80

SERIAL 1 4 6	CONT 7	A 8	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
000			IDENTIFICATION DIVISION.
010			PROGRAM-ID. SAMPLE 1410/7010 COBOL PROGRAM.
020			AUTHOR. IBM PROGRAMMING SYSTEMS.
030			REMARKS. DESIGNED TO ILLUSTRATE THE GENERAL FORM OF A COBOL-
040	-		PROGRAM.
050			
060			
070			ENVIRONMENT DIVISION.
080			
090			CONFIGURATION SECTION.
100			SOURCE-COMPUTER. IBM-1410.
110			OBJECT-COMPUTER. IBM-1410.
120			SPECIAL-NAMES. SYSTEM-OUTPUT-PRINTER IS PRINTER1
130			I-O-SWITCH EOF-SW ON STATUS IS LAST-CARD.
140			INPUT-OUTPUT SECTION.
150			FILE-CONTROL.
160			SELECT OUTPUT-PAY-FILE ASSIGN TO TAPE-UNIT MW1.
170			SELECT LIST-FILE
180			ASSIGN TO TAPE-UNIT MW1.
190			I-O-CONTROL.
200			APPLY '9' PADDING ON OUTPUT-PAY-FILE.
210			
220			

IBM

COBOL PROGRAM SHEET

Form No. X28-1464 Printed in U.S.A.

PAGE 2 3 | PROGRAM SAMPLE PROBLEM 1410/7010 COBOL | SYSTEM 1410 | SHEET 2 OF 7
002 | PROGRAMMER | DATE | IDENT 73 PAYROLL1 80

SERIAL 1 4 6	CONT 7	A 8	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
000			DATA DIVISION.
010			
020			FILE SECTION.
030			FD OUTPUT-PAY-FILE
040			BLOCK CONTAINS 10 RECORDS
050			RECORD CONTAINS 89 CHARACTERS
060			LABEL RECORDS ARE OMITTED
070			DATA RECORD IS EMPLOYEE-RECORD.
080			
090		01	EMPLOYEE-RECORD.
100			02 DISPLAY-RECORD.
110			03 FILLER SIZE IS 5.
120			03 EMPLOYEE-CODE.
130			04 MAN-NUMBER PICTURE IS 9(6).
140			04 FILLER SIZE IS 2.
150			04 MAN-NAME PICTURE IS A(20).
160			04 FILLER SIZE IS 2.
170			04 DEPT-CODE PICTURE IS 99.
180			04 FILLER SIZE IS 5.
190			04 HOURS-WORKED PICTURE IS 99.
200			03 FILLER SIZE IS 5.
210			03 CODEONE.
220			04 LABOR-GRADE PICTURE IS 99.
230			04 SHIFT PICTURE IS 9.

IBM

COBOL PROGRAM SHEET

Form No. X28-1464 Printed in U.S.A.

PAGE 1 3 | 003 | PROGRAM: SAMPLE PROBLEM 1410/7010 COBOL | SYSTEM: 1410 | SHEET 3 OF 7 | PROGRAMMER: | DATE: | IDENT. 73 PAYROLL1 80

SERIAL 4 6 7 | CONT | A 8 | B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

```
000  03 FILLER SIZE IS 5.
010  03 GROSS-PAY PICTURE IS $ZZZ.99.
020  03 FILLER SIZE IS 5.
030  03 PREMIUM-PAY PICTURE IS $ZZZ.99.
040  03 FILLER SIZE IS 5.
050  03 TOTAL-PAY PICTURE IS $ZZZ.99.
060  02 RM PICTURE IS X.
070
080  FD LIST-FILE
090  BLOCK CONTAINS 10 RECORDS
100  RECORD CONTAINS 89 CHARACTERS
110  LABEL RECORDS ARE OMITTED
120  DATA RECORD IS EMPLOYEE-RECORD1.
130
140  01 EMPLOYEE-RECORD1.
150  02 DISPLAY-RECORD1 CLASS IS AN SIZE IS 88.
160  02 RM1 PICTURE IS X.
170
180  WORKING-STORAGE SECTION.
190  77 GROSS PICTURE IS 999V99.
200  77 SHIFT-PREM PICTURE IS 999V99.
210  77 TOTAL PICTURE IS 999V99.
```

IBM

COBOL PROGRAM SHEET

Form No. X28-1464 Printed in U.S.A.

PAGE 2 3 | 004 | PROGRAM: SAMPLE PROBLEM 1410/7010 COBOL | SYSTEM: 1410 | SHEET 4 OF 7 | PROGRAMMER: | DATE: | IDENT. 73 PAYROLL2 80

SERIAL 4 6 7 | CONT | A 8 | B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

```
000  01 INPUT-RECORD.
010  02 EMPLOYEE-CODE.
020  03 MAN-NUMBER PICTURE IS 9(6).
030  03 MAN-NAME PICTURE IS A(20).
040  03 DEPT-CODE PICTURE IS 99.
050  03 FILLER SIZE IS 2.
060  03 HOURS-WORKED PICTURE IS 99.
070  02 CODEIN.
080  03 LABOR-GRADE PICTURE IS 99.
090  03 SHIFT PICTURE IS 9.
100  88 FST VALUE IS 1.
110  88 SECOND VALUE IS 2.
120  88 THIRD VALUE IS 3.
130  88 NOPREMIUM VALUE IS 4.
140
150  01 HOURLY-RATE-TABLE.
160  02 TRAINEE PICTURE IS 9V99 VALUE IS 1.50.
170  02 BEGINNER PICTURE IS 9V99 VALUE IS 1.65.
180  02 JUNIOR PICTURE IS 9V99 VALUE IS 2.00.
190  02 OPERATOR PICTURE IS 9V99 VALUE IS 2.35.
200  02 SENIOR PICTURE IS 9V99 VALUE IS 2.65.
210  02 ASSOCIATE PICTURE IS 9V99 VALUE IS 2.85.
220  02 STAFF PICTURE IS 9V99 VALUE IS 3.99.
```

IBM COBOL PROGRAM SHEET

Form No. X28-1464 Printed in U.S.A.

PAGE 005 | PROGRAM SAMPLE PROBLEM 1410/7010 COBOL | SYSTEM 1410 | SHEET 5 OF 7
PROGRAMMER | DATE | IDENT PAYROLL1

```
SERIAL  CONT  A  B
4 6     7     8  12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
000     01    RATE REDEFINES HOURLY-RATE-TABLE.
010           02 HRLY-RATE PICTURE IS 9V999 OCCURS 7 TIMES.
020
030     CONSTANT SECTION.
040           77 RMK PICTURE IS J.
050           77 FIRST-SHIFT PICTURE IS V99 VALUE IS .10.
060           77 SECOND-SHIFT PICTURE IS V99 VALUE IS .13.
070           77 THIRD-SHIFT PICTURE IS V99 VALUE IS .15.
080           77 ZERO-PREM PICTURE IS 999V99 VALUE IS 000.00.
090
100     01    HEADING-RECORD.
110           02 FILLER SIZE IS 5.
120           02 MNO PICTURE IS A(6) VALUE IS 'MAIN-NO'.
130           02 FILLER SIZE IS 5.
140           02 NM PICTURE IS A(4) VALUE IS 'NAME'.
150           02 FILLER SIZE IS 14.
160           02 DPT PICTURE IS A(4) VALUE IS 'DEPT'.
170           02 FILLER SIZE IS 3.
180           02 HRS PICTURE IS A(3) VALUE IS 'HRS'.
190           02 FILLER SIZE IS 4.
200           02 CD PICTURE IS A(4) VALUE IS 'CODE'.
210           02 FILLER SIZE IS 5.
220           02 GRSS PICTURE IS A(5) VALUE IS 'GROSS'.
```

IBM COBOL PROGRAM SHEET

Form No. X28-1464-1 Printed in U.S.A.

System 1410 | Punching Instructions | Sheet 6 of 7
Program SAMPLE PROGRAM 1410/7010 COBOL | Graphic | Card Form# | Identification PAYROLL1
Programmer | Date | Punch

```
SEQUENCE (PAGE) (SERIAL)  CONT  A  B
1 3     4 6               7     8  12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
006     000               02 FILLER SIZE IS 4.
        010               02 PREM PICTURE IS A(7) VALUE IS 'PREMIUM'.
        020               02 FILLER SIZE IS 8.
        030               02 TOTL PICTURE IS A(5) VALUE IS 'TOTAL'.
        040
        050     PROCEDURE DIVISION.
        060
        070     DECLARATIVES.
        080     UNREADABLE SECTION. USE AFTER STANDARD ERROR PROCEDURE ON
        090         LIST FILE.
        091     DISPLAY-ERROR. DISPLAY DISPLAY-RECORD1.
        100     END DECLARATIVES.
        110
        120     INTRODUCTION. NOTE THAT THIS PROGRAM HAS BEEN DESIGNED
        130         TO DEMONSTRATE TYPICAL COBOL FORMAT. NO ATTEMPT HAS BEEN
        140         MADE TO CREATE A PROGRAM FOR ACTUAL CUSTOMER APPLICATION.
        150
        160     START. OPEN OUTPUT OUTPUT-PAY-FILE.
        170     NEXT-EMPLOYEE. ACCEPT INPUT-RECORD IF LAST-CARD GO TO RWND.
        180         MOVE CORRESPONDING EMPLOYEE-CODE IN INPUT-RECORD TO EMPLOYEE-
        190         CODE IN DISPLAY-RECORD. MOVE CODEIN TO CODEOUT. MULTIPLY
        200         HOURS-WORKED IN INPUT-RECORD BY HRLY-RATE (LABOR-GRADE IN
        210         CODEIN) GIVING GROSS. MOVE GROSS TO GROSS-PAY.
        220
```

IBM

COBOL PROGRAM SHEET

Form No. X28-1464
Printed in U.S.A.

PAGE 1 3 | PROGRAM SAMPLE PROBLEM 1410/7010 COBOL | SYSTEM 1410 | SHEET 7 OF 7
00.7 | PROGRAMMER | DATE | IDENT 73 PAY ROLL 80

SERIAL 4 6 | CONT 7 | A 8 | B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

```
000  IF NOPREMIUM GO TO NO-PREM-RTN ELSE GO TO PREM1, PREM2
010  PREM3, DEPENDING ON SHIFT IN CODEIN. STOP 'INVALID PREMIUM
020- 'CODE'.
030
040  NO-PREM-RTN. MOVE ZERO-PREM TO PREMIUM-PAY. MOVE GROSS TO TOTAL-P
050- AY. GO TO OUTPUT-ROUTINE.
060  PREM1. COMPUTE SHIFT-PREM ROUNDED = GROSS * FIRST-SHIFT. GO TO
070  TOTAL-RTN.
080  PREM2. MULTIPLY GROSS BY SECOND-SHIFT GIVING SHIFT-PREM ROUNDED.
090  GO TO TOTAL-RTN.
100  PREM3. COMPUTE SHIFT-PREM ROUNDED = GROSS * THIRD-SHIFT.
110
120  TOTAL-RTN. MOVE SHIFT-PREM TO PREMIUM-PAY. ADD SHIFT-PREM TO
130  GROSS GIVING TOTAL. MOVE TOTAL TO TOTAL-PAY.
140  OUTPUT-ROUTINE. MOVE RMK TO RM. WRITE EMPLOYEE-RECORD
150  GO TO NEXT-EMPLOYEE.
160  RWND. CLOSE OUTPUT-PAY-FILE OPEN INPUT LIST-FILE.
170  NOTE PRINT OUT OF RECORD CREATED ON TAPE.
180  DISPLAY HEADING-RECORD UPON PRINTER1.
190  PRINT-RECORD. READ LIST-FILE AT END GO TO CLOSE-RTN.
200  IF CODEOUT IS EQUAL TO '999' GO TO CLOSE-RTN. DISPLAY
210  DISPLAY-RECORD1 UPON PRINTER1 GO TO PRINT-RECORD.
220  CLOSE-RTN. CLOSE LIST-FILE WITH LOCK. DISPLAY 'END OF EXECUTION'
230  STOP RUN.
```

Index

Where more than one page reference is given, major reference appears first.

Printed in U.S.A. GC28-0327-5

IBM

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
(USA Only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)