File Number GENL-24
Form C24-3235-2

IBM Systems Reference Library

COBOL (on Disk) Specifications IBM 1401, 1440, and 1460

This publication is intended for programmers who have a basic knowledge of COBOL programming. It includes the additional specifications necessary to write a COBOL program for the IBM 1401, 1440, and 1460 Data Processing Systems with disk storage.

Specific examples show how many COBOL statements are coded. A general explanation of these statements is also given.

A sample problem shows entries for all divisions.

This publication is a major revision of form C24-3235-1 and obsoletes it and prior editions. In addition to incorporating information released in Technical Newsletter N24-0293, additional information concerning programming considerations is provided.

Copies of this and other IBM publications can be obtained through IBM Branch Offices. A form is included at the back of this manual for readers' comments. If this form has been removed, address comments to: IBM Corporation, Product Publications, Dept. 245, Rochester, Minn. 55901.

Contents

Acknowledgment

In accordance with the requirements of the official government manual describing COBOL-1961 extended, 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 Material 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. DuPont de Nemours and Company
General Electric Company
General Motors Corporation
Lockheed Aircraft Corporation
The National Cash Register Company
Philco Corporation
Standard Oil Company (New Jersey)
United States Steel Corporation

"This 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 (Trade-mark of Sperry Rand Corporation), *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."

The programmer's responsibility in preparing a COBOL program is to:

1. Identify the program.
2. Specify the features and devices of the IBM 1401, 1440, or 1460 Data Processing System that will be used to compile and execute the resultant machine-language object program.
3. Describe the data to be processed.
4. State the procedure to process the data.

The programmer uses the characters, words, and expressions that make up the COBOL language. He writes them according to a standard reference format that is outlined on the COBOL program sheet (Form X28-1464). This standard coding sheet is used with all IBM COBOL systems to record the source program.

The COBOL source-program card deck is punched from these coding sheets. These cards make up the COBOL source-program card input to the COBOL processor.

Machine Requirements

To process a COBOL source program, the following minimum machine configurations are specified.

An IBM 1401 system with:
- 4,000 positions of core storage
- Advanced Programming Feature
- High-Low-Equal Compare Feature
- One IBM 1311 Disk Storage Drive with an IBM 1316 Disk Pack
- One IBM 1402 Card Read-Punch
- One IBM 1403 Printer.

An IBM 1440 system with:
- 4,000 positions of core storage
- Indexing and Store Address Register Feature
- One IBM 1311 Disk Storage Drive with an IBM 1316 Disk Pack
- One IBM 1442 Card Reader
- One IBM 1443 Printer.

An IBM 1440 system with:
- 8,000 positions of core storage
- Indexing and Store Address Register Feature
- One IBM 1301 Disk Storage
- One IBM 1442 Card Reader
- One IBM 1443 Printer

An IBM 1460 system with:
- 8,000 positions of core storage
- Indexing and Store Address Register Feature
- One IBM 1311 Disk Storage Drive with an IBM 1316 Disk Pack, or one IBM 1301 Disk Storage
- One IBM 1402 Card Read-Punch
- One IBM 1403 Printer.

The system on which the object program is to be executed must have:

1. A card reader or a disk file to load the object program resulting from the Autocoder assembly.
2. Sufficient core storage to contain the program generated by the COBOL processor. If the object program requires more than the available core-storage capacity, either the program must be executed in sections (overlays) or the job must be divided into multiple runs. This requirement is a significant consideration when planning to implement COBOL on a system with 4,000 positions of core storage.
3. The input and output devices defined in the FILE-CONTROL paragraph.
4. Sense switches if they are referred to in the SPECIAL-NAMES paragraph.
5. The expanded print-edit feature when any of the following COBOL editing functions are used:
   a. High-order CR or minus signs and high-order DB or plus signs.
   b. Floating plus and minus signs, and floating dollar signs.
   c. Check protection (asterisk fill).
   d. Decimal suppression for blank or zero fields.

COBOL Language Notation

The entire COBOL language is described in detail in the SRL publication *COBOL General Information Manual* (F28-8053). *COBOL (on Disk) Specifications for IBM 1401, 1440, and 1460* contains additional information that enables the programmer to apply the COBOL language to the IBM 1401, 1440, and 1460.

Throughout this publication, basic formats are prescribed for the various verbs, clauses, entries, and other essential elements of the COBOL language. These are generalized formats intended to guide the programmer in writing his own statements. These rules of notation must be followed:

1. All words printed entirely in capital letters are COBOL words. They have preassigned meanings in the COBOL system. For example: IDENTIFICATION DIVISION. When the COBOL processor sees these two words, it notes the beginning of the identification of the program.
2. All underlined words are required unless the portion of the format containing them is enclosed in square brackets. Square brackets [ ] indicate an optional portion of a COBOL format. Underlined words are *key words*. If any key word is missing or misspelled, it is considered an error in the program. For example:

   SEEK *file-name* RECORD

   is the COBOL format for the SEEK verb. The programmer may write either of the following entries assuming that PAYROLL is the file-name.

   SEEK PAYROLL RECORD
   SEEK PAYROLL

   SEEK is a key word and must be included. However, RECORD is an optional word and may be omitted if the user so chooses.
3. All COBOL words not underlined may be included or omitted at the option of the programmer. These words, called *optional words*, are used only for the sake of readability. Misspelling constitutes an error.
4. All lower-case words represent information that the programmer must supply. The nature of the information required is indicated. In most instances, the programmer must provide an appropriate data-name, procedure-name, or literal. For example, file-description format is

   FD *file-name*

   The programmer writes

   FD ACCOUNTS-RECEIVABLE

   ACCOUNTS-RECEIVABLE has been used as the file-name for this file-description entry.
5. Material enclosed in square backets can be used or omitted as required by the program. For example, the format for the PERFORM verb is

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

   The programmer can write one of the following statements:

   PERFORM GROSS PAY
   PERFORM GROSS PAY THRU NET PAY

   The first statement can be used to specify calculation of gross pay. The second can be used to calculate gross pay and then net pay.
6. Braces mean that one and only one of the enclosed items must be chosen. Other items are to be omitted. For example:

   LABEL RECORD[S] {ARE / IS} {STANDARD / OMITTED}

   The statement LABEL RECORDS ARE OMITTED is correct.
7. Punctuation, where shown, is essential. The programmer can insert other punctuation in accordance with the rules outlined in this publication.
8. Special characters, such as the equal sign, are essential where shown, although they may not be underlined.
9. In certain cases, a succession of operands or other elements may be used in the same statement. In such a case, the 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 is enclosed within brackets and the closing bracket is followed by three dots, the entire group (not merely the last operand) must be repeated if any repetition is required.
10. Restrictions and comments on each basic format will be found in this publication. The formats should not be used without reading the accompanying text.

The COBOL source program has four major divisions. Each division has its own set of statements, which are written according to the rules established for the COBOL language, as described in the *IBM COBOL General Information Manual* (F28-8053). These division-statement sets must be arranged for presentation to the 1401, 1440, and 1460 COBOL processor in this order:

IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
PROCEDURE DIVISION.

The IDENTIFICATION DIVISION entries are written as described in the IBM COBOL General Information Manual.

Environment Division

In this part of the COBOL source program, the programmer specifies the physical characteristics of the particular IBM 1401, 1440, and/or 1460 system(s) to be used to compile and to execute the object program.

The ENVIRONMENT DIVISION has two major sections, each of which has a fixed section name: CONFIGURATION and INPUT-OUTPUT.

The 1401, 1440, and 1460 COBOL presentation format for this is:

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

Configuration Section

The CONFIGURATION section has three paragraphs. The SOURCE-COMPUTER paragraph names the system that will compile the object program from the COBOL source statements.

The OBJECT-COMPUTER paragraph names and describes the system that will execute the object program.

The SPECIAL-NAMES paragraph equates: mnemonic names to standard names for actual machine devices, condition-names to standard names for the status of actual machine switches, and Autocoder-names to COBOL-names.

Source-Computer Paragraph

Reference Format

```
SOURCE-COMPUTER.   { IBM-1401 }
                   { IBM-1440 }.
                   { IBM-1460 }
```

This statement is required in all 1401, 1440, and 1460 COBOL source programs.

Object-Computer Paragraph

Reference Format

```
OBJECT-COMPUTER.   { IBM-1401 }
                   { IBM-1440 }
                   { IBM-1460 }

[ MEMORY SIZE { { 4000  }                          }
              { { 8000  } CHARACTERS               }
              { { 12000 }                          }
              { { 16000 }                          }
              { ADDRESS integer { THROUGH }        }
              {                 { THRU    }        }
              {   { 4000  }                        }
              {   { 8000  }                        }
              {   { 12000 }                        }
              {   { 16000 }                        } ]

[NO-PRINT-STORAGE]
[NO-MULTIPLY-DIVIDE]
[NO-DIRECT-SEEK]
[NO-OVERLAP]
[NO-CONSOLE-PRINTER].
```

The OBJECT-COMPUTER paragraph describes the computer that will execute the object program. The OBJECT-COMPUTER. IBM-1401 (or 1440 or 1460) statement without optional clauses defines an IBM 1401 (or 1440 or 1460) with 16,000 positions of core storage, the processing overlap feature (1401 and 1460 systems only), the input/output units required for the files defined in the FILE-CONTROL paragraph, the direct-seek feature, the multiply/divide feature, and print storage. If the object machine has fewer than 16,000 positions of core

storage, and/or if any of these features are not present in the object machine, the appropriate clause must be included in the source program.

```
[ MEMORY SIZE { { 4000 } }                                  ]
              { { 8000 } } CHARACTERS                       ]
              { { 12000} }                                  ]
              { { 16000} }                                  ]
              { ADDRESS integer { THROUGH } { 4000 } }      ]
              {                 { THRU    } { 8000 } }      ]
              {                           { 12000} }      ]
              {                           { 16000} }      ]
```

This clause tells the processor how many positions of core storage are available in the object machine and the starting core-storage address of the object program.

If the programmer wishes the program to start at any location other than 334, and if a printer is not to be used as an output device, he can use the ADDRESS *integer* THRU option and write the numerical address of this location in the *integer* portion. This number should not be less than 334. If a printer is to be used as an output device, the program starts at location 469. If the programmer wishes the program to start at any location other than 469, and if a printer is to be used as an output device, the *integer* portion of the ADDRESS *integer* THRU option must be greater than 469.

If the MEMORY SIZE statement is omitted from the COBOL source program, the processor assumes that the object computer has 16,000 positions and starts the object program at core-storage location 334.

```
[NO-PRINT-STORAGE]
[NO-MULTIPLY-DIVIDE]
[NO-DIRECT-SEEK]
[NO-OVERLAP]
[NO-CONSOLE-PRINTER].
```

These clauses tell the processor that the object machine is not equipped with certain special features.

If either NO-PRINT-STORAGE or NO-DIRECT-SEEK is specified, the IOCS generated instructions will not use those machine features.

If NO-MULTIPLY-DIVIDE is specified, a subroutine will be included in and used by the object program whenever COMPUTE is used with *, /, or **, or whenever MULTIPLY or DIVIDE is used. If the NO-MULTIPLY-DIVIDE clause is not specified, the multiply/divide special feature will be used by the object program.

The NO-OVERLAP option must be included only if the object computer is an IBM 1401 or IBM 1460 that does not have the processing overlap feature.

If the system is 1440 or 1460, and if an IBM 1447 is included in the system, the NO-CONSOLE-PRINTER option will cause a STOP-literal statement to display the literal itself or its address in the B-address register. If this clause is omitted, the literal will be displayed on the console printer.

Special-Names Paragraph

Reference Format

```
SPECIAL-NAMES.

[ device-name  IS mnemonic-name
              [ device-name  IS mnemonic-name ] ].

[ switch-name { ON  } STATUS IS condition-name
              { OFF }
              [ { OFF } STATUS IS condition-name ]
              [ { ON  }                         ]
              [ switch-name . . . ] ].

[ autocoder-name IS cobol-name
              [ autocoder-name . . ] ].
```

This paragraph equates: mnemonic names to the standard names for actual machine devices, condition-names to the status of actual machine switches, and Autocoder-names to COBOL-names.

Device-Names

The standard device-names for the IBM 1401, 1440, and 1460 systems indicate to the COBOL processor which devices are available in the object computer. They are written with the mnemonic-name the programmer has used to refer to them in the PROCEDURE DIVISION. This is a list of device-names:

Device-Name	*Actual Device*
1402-R, *n*	1402 Card Reader
1442-R, *n*	1442 Card Reader
1402-P, *n*	1402 Card Punch
1442-P, *n*	1442 Card Punch
1444-P	1444 Card Punch
1403-P	1403 Printer
1443-P	1443 Printer
1403-CT, *n*	1403 Carriage Tape
1443-CT, *n*	1443 Carriage Tape
1447-CP	1447 Console Printer

1401 and 1460 Device-Names. For the 1402-R and 1402-P device-names, *n* is a digit specifying the stacker into which a card is to fall. For the card reader it must be a 0 (normal read), 1 (read select), or 2 (common). For the card punch it must be 0 (normal punch), 4

(punch select), or 8 (common). If one of the digits is not included with a 1402 device-name, the processor assumes that the stacker desired is 1 for a read operation and 4 for a punch operation. If *n* is coded, there must be a space between it and the device-name as in 1402-R, 1.

1440 Device-Names. For the 1442-R and 1442-P, *n* is the digit (1 or 2) that specifies the unit (1442 Unit 1 or 1442 Unit 2) of the card read-punch to which the device-name is assigned. If *n* is not specified, the processor will assume Unit 1. If *n* is coded, there must be a space between it and the device-name as in 1442-R, 1. 1444-P refers to the 1444 card punch unit 3.

For all systems, the printer is the assumed standard output unit for use with the DISPLAY verb. The card reader is the standard input device for use with the ACCEPT verb. However, if the IBM 1447 Console Printer is equated with a special name, that unit may be specified as an input or output unit (or both) with the ACCEPT and DISPLAY verbs.

For the carriage tape device-name, *n* specifies which channel in the carriage tape terminates a particular carriage skip. It can be any number from 1 to 12. This name is used with the ADVANCING option of the WRITE verb (see *Procedure Division*). If *n* is not coded, the processor assumes that the skip is to channel 1. If *n* is coded, there must be a space between it and the device-name as in 1443-CT, 3 or 1403-CT, 3.

Note: Punched-card input and output devices should not be used with both the DISPLAY and WRITE verbs in the same program. The same restriction applies to using these devices with both the ACCEPT and READ verbs.

$$\left[\quad \textit{autocoder-name}\ \underline{\text{IS}}\quad \textit{cobol-name}\right]$$

General Description: This statement enables the programmer to write Autocoder statements that refer to COBOL data-names and procedure-names (see ENTER).

If an Autocoder-name is used to refer to an area that has been defined by a COBOL statement, the COBOL name must be equated to the Autocoder-name.

Example: If TOTALS is a COBOL-name used to define a COBOL area and the symbol TOTLS is used in an Autocoder statement to refer to the same area, the statement shown in Figure 1 must appear in the SPECIAL-NAMES paragraph of the COBOL program.

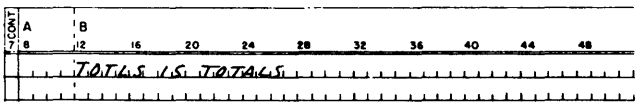


Figure 1. Equating an Autocoder-Name to a COBOL-Name

A symbol used as an Autocoder-name must meet these requirements:

1. It must be five characters long.
2. It must begin with an alphabetic character.
3. It cannot contain a special character.
4. A blank cannot appear within the symbol.

The COBOL-name must be a non-qualified procedure-name or data-name. It cannot be a condition-name.

Switch-Names and Conditions

General Description: A switch-name is written followed by the condition-names used to identify ON STATUS and OFF STATUS.

The standard switch-names are:

Switch-Name	*Indicates*
1403-P-CB	Printer Carriage Busy
1443-P-CB	Printer Carriage Busy
1403-P-C9	Sense Carriage Tape Channel 9
1443-P-C9	Sense Carriage Tape Channel 9
1403-P-CV	Sense Carriage Tape Channel 12 (Overflow)
1443-P-CV	Sense Carriage Tape Channel 12 (Overflow)
1460-SS *x*	Sense Switch
1401-SS *x*	Sense Switch
1440-SS *x*	Sense Switch

The *x* in the sense switch is the actual letter that represents a specific 1401, 1440, or 1460 sense switch. This must be a letter within the range A-G. There must be at least one space between the switch designation and the letter used for *x*. For example, 1401-SS G.

The status of 1401, 1440, and 1460 switches may be interrogated by expressions in the PROCEDURE DIVISION that use condition-names.

Examples: Figures 2 and 3 show examples of SPECIAL-NAMES paragraphs.

Input-Output Section

The INPUT-OUTPUT section has two paragraphs.

The FILE-CONTROL paragraph names each file, identifies its input or output medium, and assigns it to one or more input/output devices.

File-Control Paragraph

Reference Format

$$\underline{\text{FILE-CONTROL}}.\ \underline{\text{SELECT}}\ \textit{file-name-1}$$

$$\underline{\text{ASSIGN}}\ \text{TO}\ \textit{device-name}$$

$$\left[\underline{\text{RESERVE}}\ \left\{\begin{matrix}1\\ \underline{\text{NO}}\end{matrix}\right\}\ \text{ALTERNATE AREA[S]}\right].$$

$$\left[\underline{\text{SELECT}}\ \ldots\ldots\ldots\right].$$

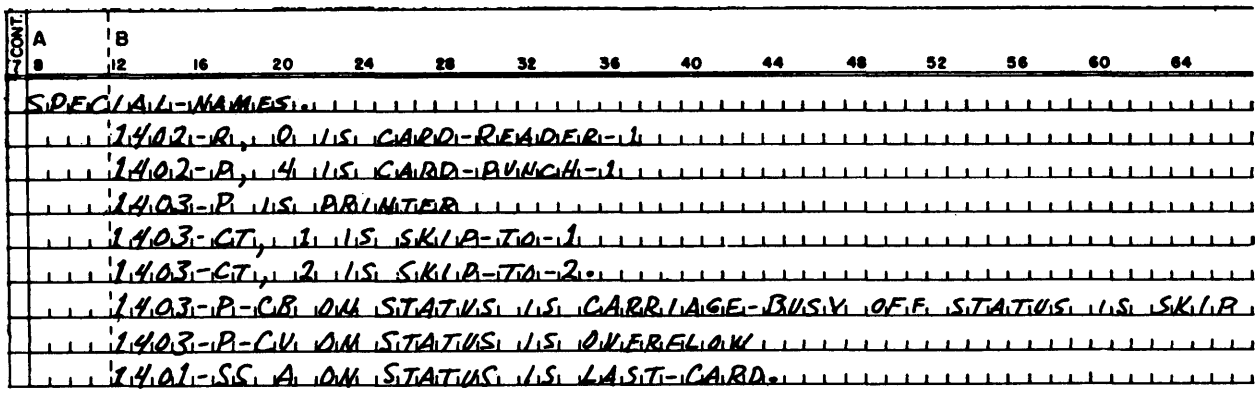


Figure 2. 1401 and 1460 SPECIAL-NAMES Paragraph

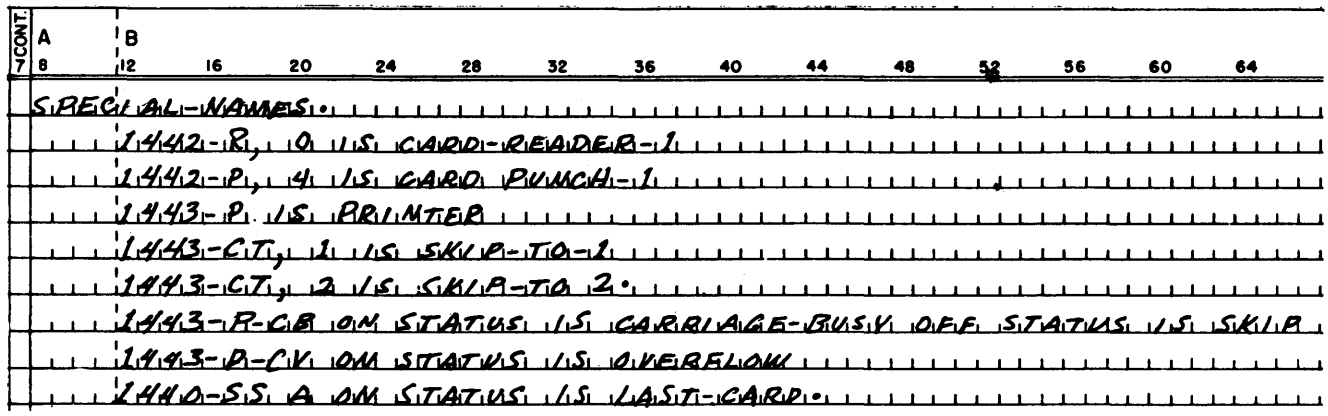


Figure 3. 1440 SPECIAL-NAMES Paragraph

This paragraph names each file used in the source program, identifies its media, and assigns it to an input or output device. It also permits the programmer of 1401 and 1460 systems to specify an alternate input/output area for magnetic tape files if the 1401 or 1460 has the processing-overlap feature.

SELECT *file-name-1*

Each file to be processed by the READ or WRITE verbs in the PROCEDURE DIVISION must be named in a SELECT file-name entry. This file-name must be unique within the source program and must be described by a file-description entry in the DATA DIVISION of the source program.

Example: Figure 4 shows a sample SELECT file-name entry.

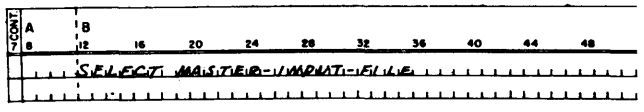


Figure 4. SELECT

ASSIGN TO *device-name-1*

This clause is used to assign a file to an input or output device-name. The device-names that are valid in the FILE-CONTROL paragraph are:

Device-Name	*Actual Device*
1402-R, *n*	1402 Card Reader
1442-R, *n*	1442 Card Reader
1402-P, *n*	1402 Card Punch
1442-P, *n*	1442 Card Punch
1444-P	1444 Card Punch
1403-P	1403 Printer
1443-P	1443 Printer
1311-D, *d*	1311 Disk Storage Drive
1301-D, *d*	1301 Disk Storage
TAPE(S) *u, a*	729 or 7330 or 7335 Magnetic Tape Unit

Punched-Card Device-Names

The punched-card devices that are valid in the FILE-CONTROL paragraph are the card reader, the card punch, and the printer.

For the 1402-R and 1402-P device-names, *n* is a digit specifying the stacker into which a card is to fall. For the card reader it must be a 0 (normal read), 1 (read select), or 2 (common). For the card punch it

must be 0 (normal punch), 4 (punch select), or 8 (common). If one of the digits is not included with a 1402 device-name, the processor assumes that the stacker desired is 1 for a read operation and 4 for a punch operation. If *n* is coded, there must be a space between it and the device-name as in 1402-R, 1.

For the 1442-R and 1442-P device-names, *n* is a digit (1 or 2) specifying the unit in which a file is to be placed. If *n* is not coded, the processor assumes unit 1. If *n* is coded, there must be a space between it and the device-name as in 1442-R, 1.

Disk-Storage Device-Names

For disk files, 1311-D or 1301-D is the device-name. It indicates that the file is to be assigned to either a 1301 or a 1311. For both the 1301 and 1311, the *d* specifies the particular unit and can be any digit from 0 to 4. There must be a space between the comma and the digit.

Example: Figure 5 shows an ASSIGN statement for a 1311 disk file.

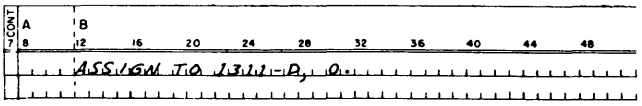


Figure 5. ASSIGN Disk-File

Magnetic-Tape Device-Names

For magnetic-tape files, TAPE(S) is the device-name. It indicates that the file is to be assigned to a tape unit. The *u* specifies the particular unit to be assigned. It can be any digit from 1 to 6 with the 1401 or 1460. For the 1440, *u* can be 1 or 2. The *a* specifies that an alternate unit is to be assigned. It can be any digit from 1 to 6 with the 1401 or 1460, but should not be the same digit that is used for *u*. For the 1440, *a* can be 1 or 2, but should not be the same digit that is used for *u*. The same tape unit (or alternate tape unit) should not be assigned more than once in a COBOL source program. There must be a space between *u* and *a*.

Example: Figure 6 shows an ASSIGN statement for a tape file.

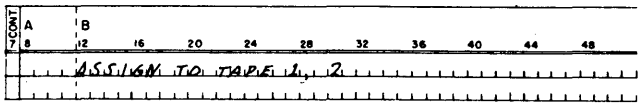


Figure 6. ASSIGN Tape-File

$$\left[\underline{\text{RESERVE}}\ \left\{\begin{matrix}1\\ \underline{\text{NO}}\end{matrix}\right\}\ \text{ALTERNATE AREA[S]}\right].$$

This statement reserves one or no alternate area for a magnetic-tape file. One alternate area may be specified only if the object machine has the processing overlap feature. If this clause is missing from the source program, the processor reserves no alternate area.

Example: Figure 7 shows a sample RESERVE statement.

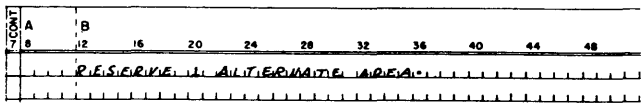


Figure 7. RESERVE

Note. In the SPECIAL-NAMES paragraph and the FILE-CONTROL paragraph, two unit-record files cannot be assigned to the same device. For example, if the punch is defined in the SPECIAL-NAMES paragraph, it cannot be defined again in the SPECIAL-NAMES paragraph, nor can it be defined in the FILE-CONTROL paragraph.

I-O-Control Paragraph

Reference Format

I-O-CONTROL.

$$\underline{\text{APPLY}}\ \left\{\begin{matrix}\underline{\text{TYPE-A-LABEL}}\\ \underline{\text{TYPE-B-LABEL}}\\ \underline{\text{TYPE-C-LABEL}}\end{matrix}\right\}\ \underline{\text{ON}}\ \textit{file-name}\ \left[\text{APPLY}\ldots\right].$$

The I-O-CONTROL paragraph is used to specify the type of the label records for tape files.

Type-A, -B, and -C label records contain 120, 80, and 84 characters, respectively. The file-name refers to the file-name assigned to the file in the associated FD entry.

Example: Figure 8 shows a sample APPLY statement.

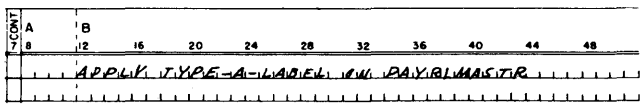


Figure 8. APPLY

Deferred Elements of the Environment Division

Several elements described in the COBOL General Information Manual are not contained in this COBOL processor. These should not be coded in the ENVIRONMENT DIVISION entries for a 1401, 1440, or 1460 COBOL program. They are stated here for reference.

1. The OPTIONAL option of the FILE-CONTROL paragraph.
2. The MULTIPLE REEL option in the FILE-CONTROL paragraph and all other features that provide for automatic assignments of tape units for a file.
3. The RENAMING option of the FILE-CONTROL paragraph.

4. The entire COPY option. (The library tape for the 1401 COBOL processor does not presently support the copy feature.)
4. The RERUN option of the I-O-CONTROL paragraph.

Not Applicable

The ASSIGN option of the OBJECT-COMPUTER paragraph.

Data Division

Each file, record, and data item is described within a program by writing data-description entries in the source program. Every data-name referred to in the PROCEDURE DIVISION except figurative constants must be described in the DATA DIVISION. Items and records are described by *record-description entries*, and files are described by *file-description entries* (MD and FD entries).

Detailed information about record formats is presented in the SRL publications *Input/Output Control System (on Disk) for IBM 1401/1460: Specifications* (C24-1489) and *Input/Output Control System for IBM 1440: Specifications* (C24-3011). General information is presented in the following sections.

Record Formats for Tape Files

Form-1 Records

Form-1 tape records are fixed length, unblocked, with or without record marks. *Fixed-length* implies that all records in the file have the same number of characters. *Unblocked* means that one data record is contained in one tape record. A record mark (+) is a special character written at the end of a data record to indicate that the preceding character is the last record character. If input records are form-1 but are to be written as output in form-2 or form-4, they should have record marks. Otherwise the use of record marks is optional. Tape records are physically separated by a section of blank tape called an Interrecord Gap (IRG). Figures 9 and 10 show examples of form-1 records with and without record marks.

Form-2 Records

Form-2 records are fixed length, blocked, with record marks, and with padding of short-length blocks. *Blocked* means that more than one data record is contained in one tape record (two or more data records occupy the space between two interrecord gaps). Record marks must be used to separate the data records.

Padding means that nines (9's) are used to fill the last block for a file if there are not sufficient *data* records to fill it. Thus, a fixed-length block will always contain the same number of characters, but a padded record(s) will be substituted if there are not enough data records to fill the last block.

Figure 11 shows fixed-length, blocked tape records with record marks and padding. Each block contains four records.

Form-3 Records

Form-3 records (variable unblocked) are not permitted with COBOL.

Form-4 Records

Form-4 tape records are variable-length, blocked, with record marks and a Record Character-Count (RCC) field in each record, and a Block Character-Count (BCC) field in each block. *Variable length* implies that all the records in a file do not contain the same number of characters.

Block Character-Count Field

A four-character field at the beginning of each block contains a count of the total number of characters in the block (including the block character-count field itself). The BCC field has AB zone bits (IBM card code 12-punch) over the units position. This count is used to check wrong-length record conditions.

Record Character-Count Field

A record character-count field of three characters in each record contains a count of the number of charac-

Figure 9. Form-1 Records with Record Marks

Figure 10. Form-1 Records without Record Marks

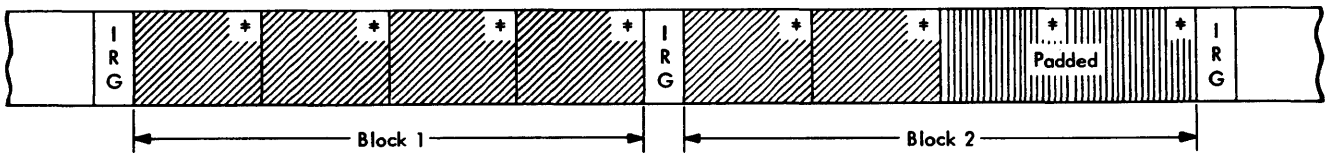


Figure 11. Form-2 Records with Padding

ters in that record, including the RCC field itself and the record mark. This field must be in the same relative position in each record (the character size of each C1 in Figure 12 is the same). Figure 12 shows the record format for a form-4 record.

Note: For form-2 and form-4 records, it is the programmer's responsibility to place all record marks in the file-description entries, and in the work areas, where applicable.

Record Formats for Punched-Card Files

Card Read-Punch Records

Records of files assigned to the card reader and the card punch must be eighty characters long, unblocked, and may or may not have record marks in the 80th character position (card column 80). This is equivalent to the form-1 record described previously.

Printer Records

Records of files assigned to the printer must also have form-1 record format. For the printer the fixed record size must be equal to the number of print positions on the printer. A maximum of 132 print positions is used by the COBOL compiler.

Record Formats for Disk Files

COBOL can process disk records that are fixed-length unblocked (form-1), fixed-length blocked (form-2), or variable-length blocked records (form-4). The maximum size of a record is 999 characters. Figure 22 shows the record forms permitted for each type of access mode.

To process blocked records, the COBOL processor requires the following.

1. A block may contain a maximum of ten records for random files, one hundred for sequential files, and thirty for control-sequential files.
2. In blocked files, each record in every block must contain a record mark as its last character.
3. For variable-length records, a block-length field must be included in each block, and a record-length field in each record (see Figure 13).

   As the name implies, *block length* is the total number of characters in the block, including itself and record marks. The block-length field must always be recorded in the first four positions of the block. When output records are created by COBOL, this count is generated automatically.

Record length is the total number of characters in the record, including itself and the record mark. The record-length field is a three-position field and must be located in the same three positions within each record in the file.

Figure 13 shows examples of the various types of disk records that this COBOL processor can handle.

Data Division Language Specifications

The DATA DIVISION of a COBOL source program is divided into three major sections:

FILE SECTION.
WORKING-STORAGE SECTION.
CONSTANT SECTION.

The FILE SECTION describes the input and output files with respect to content and organizational format. It has two major subdivisions: the file-description entry that specifies the physical characteristics and organization of the input and/or output data and the record-description entry that describes the individual items contained in the file records.

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

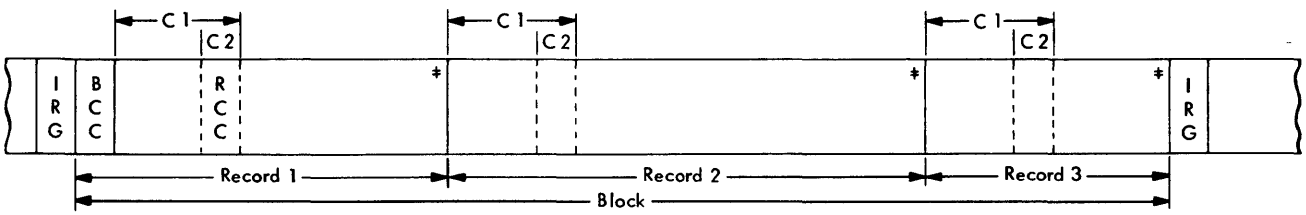


Figure 12. Form-4 Records

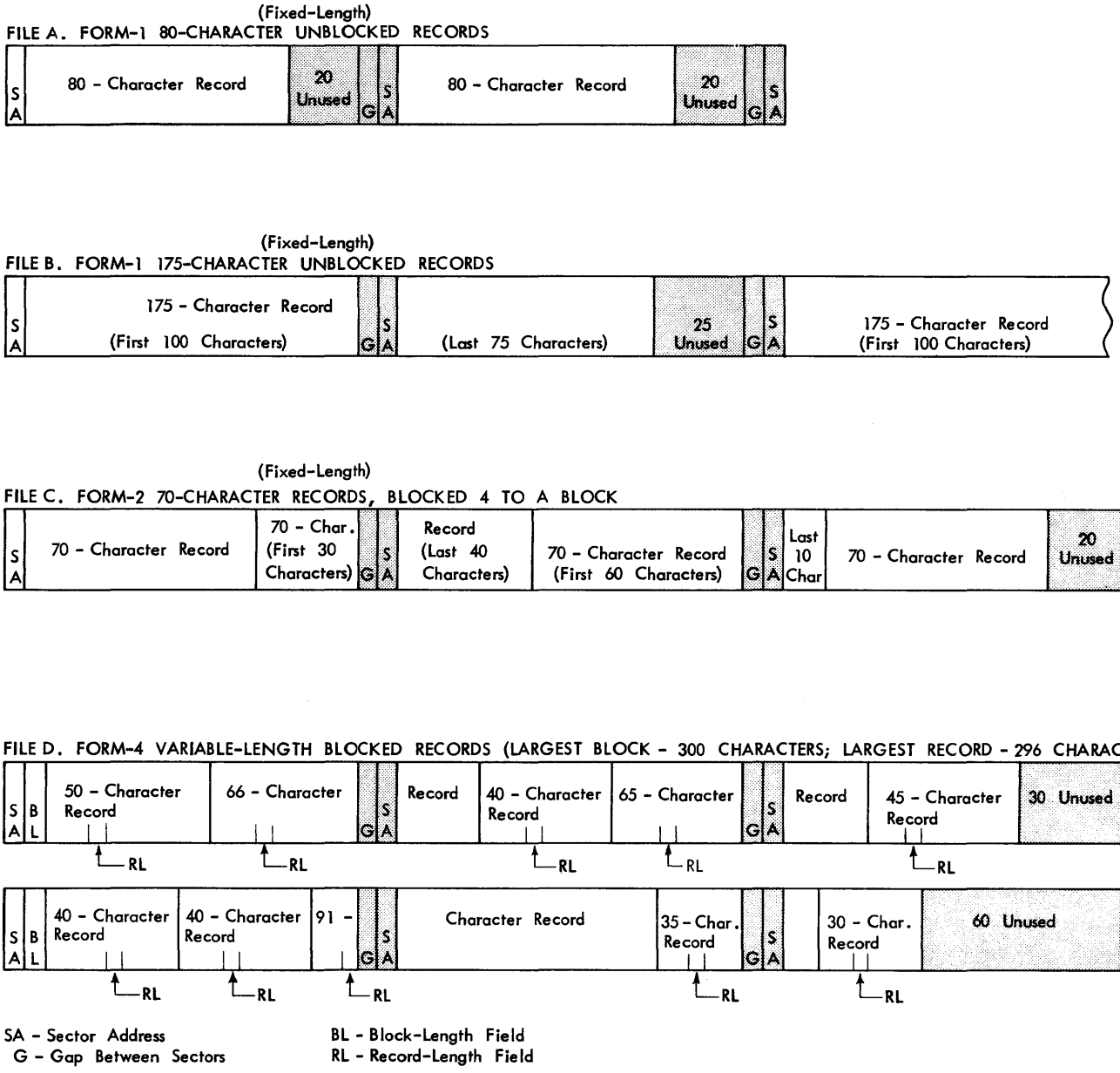


Figure 13. Schematic Records on Disk

object program. A date, for example, might be a fixed item, or a constant.

The COBOL presentation format for the DATA DIVISION is:

DATA DIVISION.

FILE SECTION.
File-Description Entries and
Record-Description Entries

WORKING-STORAGE SECTION.
Record-Description Entries

CONSTANT SECTION.
Record-Description Entries

File Section

The file-description entries and record-description entries describe the files to be processed by the object program. The file-description entries are of two major types: those that involve the disk-storage unit and those that involve other input or output media.

File-Description Entries

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

File-Description Entry—Tape Files

This format is used to describe magnetic tape files.

Reference Format

```
FD file-name [ RECORDING MODE IS mode ]

[ BLOCK CONTAINS integer-1 { RECORD[S] } ]
                           { CHARACTER[S] }

[ RECORD CONTAINS [integer-2 TO]
                  integer-3 CHARACTER[S] ]

LABEL RECORD[S] { ARE } { STANDARD }
                { IS  } { OMITTED  }

[ VALUE OF data-name-1 IS literal [ data-name-2 IS ... ] ]

DATA RECORD[S] { ARE } data-name-3 [ data-name-4 ].
               { IS  }
```

Note: A VALUE clause is required when LABEL RECORDS are standard.

FD *file-name*

The level indicator identifies the beginning of the file-description entry and precedes the file-name assigned by the programmer. (*Example:* Figure 14.)

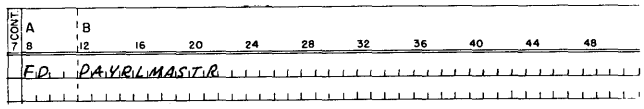


Figure 14. FD File-Name

[RECORDING MODE IS 1]

This clause specifies the mode in which the file is recorded. A *1* indicates the move mode, even-parity. RECORDING MODE *1* is the only recording mode implemented by this COBOL processor.

[BLOCK CONTAINS *integer-1* { RECORD[S] }]
                             { CHARACTER[S] }

This clause must be included if more than one data record is included in a tape record (other than form-1). It indicates the size of the block in records or characters. The size may be stated in terms of RECORD(s) for form-1 or form-2 records where *integer-1* is the number of data records in the block.

The size must be stated in terms of CHARACTER(s) for form-4 records where *integer-1* is equal to or greater than the number of characters in the longest block of the file. This number includes the four-character block count field (BCC). See also *Form-4 Records.*

Example: The largest block in the PAYRLMASTR file contains 500 characters plus the BCC field (Figure 15).

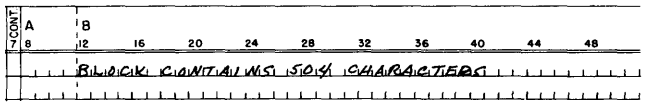


Figure 15. BLOCK CONTAINS

[RECORD CONTAINS [*integer-2* TO]
                  integer-3 CHARACTER[S]]

The RECORD CONTAINS clause may be used to specify the number of characters in the data records. Because the record-description entries define the size of each data record, this clause is never necessary. However, if the programmer wishes to include it, *integer-2* specifies the number of characters in the smallest record in the file, and *integer-3* specifies the number of characters in the largest record.

Fixed-length records must be specified using *integer-3* only. Variable-length records are specified by using both *integer-2* and *integer-3*.

Example: The records for a certain file are variable length. The smallest record size is 75 characters; the largest is 86 characters (Figure 16). When levels are defined in the FD, 86 must be defined before 75.

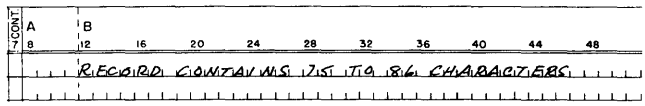


Figure 16. RECORD CONTAINS

Note: Both the block count and the record count must include the record mark (‡) in the count.

LABEL RECORD[S] { ARE } { STANDARD }
                { IS } { OMITTED }

This required clause states whether header and trailer label records are standard or omitted. This COBOL processor can handle only standard type-A, type-B, and type-C tape labels and standard disk labels. For punched-card files the OMITTED option must be used.

Example: Figure 17 shows a LABEL RECORD entry for a punched-card input file.

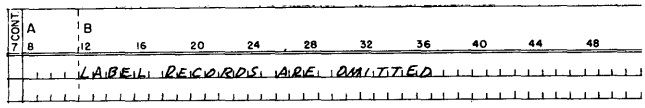


Figure 17. LABEL RECORDS

Today's Date

If standard label records are specified for output files, today's date must be in core storage at object-program execution time.

To enter the current date in the object program, insert a date card just ahead of the EX card produced by the Autocoder processor. The EX card is the last card in the object program. The format for the date card is:

Columns	*Punch*	*Description*
1-3	082	Storage Location
4-5	05	Number of Characters
6	0-5-8	Word Separator
7-11	xx xxx YR DAY	Today's Date

```
[ VALUE OF data-name-1 IS literal    [ data-name-2 IS ... ] ]
```

The COBOL programmer may specify the items of information that appear in the label records of tape files. These items must be supplied by using a VALUE OF clause if standard tape header-label records are used.

Data-name is the name of a field contained in the header label record; literal refers to the contents of the field. Figure 18 is a chart showing data-names and lengths of fields used in standard tape header label records.

Label Information (Header Label Records)

The 22-character label-information field in 120-character label records contains these fields:

Field Name	*Number of Characters*	*Type of Characters*
Density	1	Numeric
Check Sum	1	Numeric
Block Sequence	1	Numeric
Tape Checking Technique	1	Numeric
Tape Data Recording Technique	1	Numeric
Tape Data Processing Technique	1	Numeric
Creating System	4	Numeric (1401, 1440, or 1460)
Record Format	1	Alphanumeric
Record Length	5	Numeric
Blocking Factor/Size	5	Numeric
Check Point	1	Numeric

The 6-character label information field in 84-character label records contains a blank and five numeric characters.

Tape Trailer Labels

The following information is contained in IBM standard trailer labels:

	Positions	*Contents*
TYPE-A-LABELS (120 characters)	1-5	1 EOR b 1 EOF b
	67-72	XXXXXX (Block Count)
TYPE-B-LABELS (80 characters)	1-5	1 EOR b 1 EOF b
	6-10	XXXXX (Block Count)
TYPE-C-LABELS (84 characters)	1-6	1 EOR bb 1 EOF bb
	7-12	XXXXXX (Block Count)

LABEL RECORD FIELD	COMPLETE CHECKING INPUT A	B	C	COMPLETE CHECKING OUTPUT A	B	C	PARTIAL CHECKING INPUT A	B	C	PARTIAL CHECKING OUTPUT A	B	C
IDENTIFICATION (or ID)	10 A/N	10 A/N	18 A/N	10 A/N	10 A/N	18 A/N	10 A/N	10 A/N	18 A/N	10 A/N	10 A/N	18 A/N
CREATION-DATE	5 N	5 N	5 N									
** RETENTION-CYCLE	4 N	3 N	3 N	4 N	3 N	3 N				4 N	3 N	3 N
*** FILE-SERIAL-NUMBER	5 N	5 N	5 N	5 N	5 N	5 N						
** REEL-SEQUENCE-NUMBER	†4 N	†3 N	†4 N	†4 N	†3 N	†4 N						
LABEL-INFORMATION	* 22 A/N		Δ6 A/N	* 22 A/N		Δ6 A/N						

† If not present, 001 or 0001 will be assumed.
* All 22 characters are checked.
** If the label type requires only 3 digits, the thousands position must be zero.
*** The use of FILE-SERIAL-NUMBER implies full label checking for this file.
Δ Optional but checked.

Figure 18. Header Label Records for Tape Files

Figure 19 shows how the identification of a payroll master file is supplied.

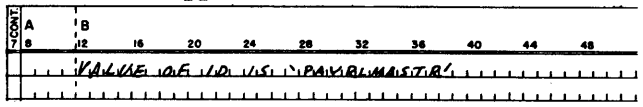


Figure 19. IDENTIFICATION

Examples: Figure 20 shows how IDENTIFICATION and a retention cycle of 286 days are supplied for an output file.

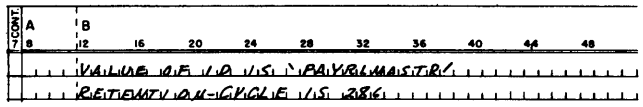


Figure 20. IDENTIFICATION and Retention Cycle

DATA RECORD[S] {ARE / IS} *data-name-3* [*data-name-4*].

Data-name-3, data-name-4, etc., must each be the subject of a record-description entry that has a level-number of 01.

If the file contains more than one type of record, a different data-name must appear for each type. Data-name order is not important.

Note: If one record is read from a given file and another is read from the same file, the second record replaces the first in the read-in area. Thus, if two records are needed for processing at the same time, the first record must be saved by moving it to another area of storage (such as a work area) before the second record is read.

Examples: Figure 21 shows a sample DATA RECORD clause. In this example, RECORDA and RECORDB are both records in the same file and are described in a record-description entry as level 01 records.

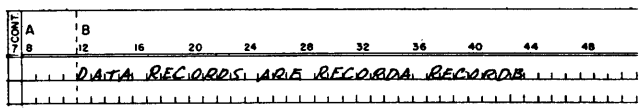


Figure 21. DATA RECORDS

File-Description Entry—Punched-Card Files

This format is used to describe punched-card files.

FD *file-name* [RECORDING MODE IS 1]

[BLOCK CONTAINS *integer-1* {RECORD[S] / CHARACTER[S]}]

[RECORD CONTAINS *integer-3* CHARACTER[S]]

LABEL RECORD[S] {IS / ARE} OMITTED

DATA RECORD[S] {ARE / IS} *data-name-3* [*data-name-4*].

Mass Storage Files

With this COBOL processor, the term *mass-storage file* refers to any group of records read from, stored on, or written on a disk storage unit.

Three reference formats exist that allow the user to specify random processing, control-sequential processing, or sequential processing (see *Access Modes*).

In any one COBOL source program, a maximum of seven MD entries can be used.

Random Access

Reference Format

MD *file-name*

[RECORDING MODE IS SECTOR]

PROCESSING MODE IS SEQUENTIAL

ACCESS MODE IS RANDOM

ACTUAL KEY IS *data-name*

[SYMBOLIC KEY IS *data-name*]

FILE-LIMIT[S] {IS / ARE} *integer* {THRU / THROUGH} *integer*

[BLOCK CONTAINS *integer* {RECORD[S] / CHARACTER[S]}]

[RECORD CONTAINS *integer* CHARACTER[S]]

LABEL RECORD[S] {IS / ARE} {STANDARD / OMITTED}

[VALUE OF *data-name* IS *literal* [*data-name* IS . . .]]

DATA RECORD[S] {IS / ARE} *data-name* [*data-name* . . .]

Control-Sequential Access

Reference Format

```
MD file-name

[RECORDING MODE IS SECTOR]

    PROCESSING MODE IS SEQUENTIAL
    ACCESS MODE IS { CONTROL-SEQUENTIAL-A }
                   { CONTROL-SEQUENTIAL-B }

    FILE-LIMIT[S] {IS}  integer {THRU   }  integer
                  {ARE}         {THROUGH}

[BLOCK CONTAINS integer {RECORD[S]   }]
                        {CHARACTER[S]}

[RECORD CONTAINS integer CHARACTER[S]]

    LABEL RECORD[S] {IS}  {STANDARD}
                    {ARE} {OMITTED }

[VALUE OF data-name IS literal [data-name IS ...]]

    DATA RECORD[S] {IS}  data-name [data-name ...] .
                   {ARE}
```

Sequential Access

```
MD file-name

[RECORDING MODE IS SECTOR]
 PROCESSING MODE IS SEQUENTIAL
 ACCESS MODE IS SEQUENTIAL
 FILE-LIMIT[S] {IS}  integer {THRU   }  integer
               {ARE}         {THROUGH}

[BLOCK CONTAINS integer {RECORD[S]   }]
                       {CHARACTER[S]}

[RECORD CONTAINS [integer TO]
                 integer CHARACTER[S]]

   LABEL RECORD[S] {IS}  {STANDARD}
                   {ARE} {OMITTED }

[VALUE OF data-name IS literal [data-name IS ...]]

   DATA RECORD[S] {IS}  data-name [data-name ...] .
                  {ARE}
```

Clause Description

The file-description entry clauses that have not been described previously are described here.

```
[RECORDING MODE IS SECTOR]
```

A recording mode is significant only for mass-storage files. All files will be recorded and read without word marks. The key word SECTOR is used to specify the particular disk-recording mode for a specific file. Data is read or written by sector (100 characters per sector).

```
PROCESSING MODE IS SEQUENTIAL
```

This clause is used to describe all mass-storage files for the IBM 1401, 1440, or 1460 Data Processing Systems. The clause refers to the order of action of the access mechanisms. Because the 1311 has only one access mechanism, the access order is sequential. (Record 1 is retrieved, followed by record 2, etc., as called for by the object program.)

```
                 { RANDOM              }
ACCESS MODE IS   { CONTROL-SEQUENTIAL-A }
                 { CONTROL-SEQUENTIAL-B }
                 { SEQUENTIAL          }
```

This clause must be included in an MD entry and specifies the organization of the file. The terms RANDOM, CONTROL-SEQUENTIAL, and SEQUENTIAL apply to the means of retrieving and storing disk records.

RANDOM

In the RANDOM access mode, each record has a unique disk address. Records can be arranged in random order because the unique address can refer to only one record in the file. In a purely random file, one seek and one read per record is necessary.

SEQUENTIAL

In the SEQUENTIAL access mode, records are arranged in sequence by control field. They are stored in consecutive locations in the disk-storage unit between addresses established by the user as the upper and lower limits of the file area. To process a sequential file, the program requires only the upper and lower limits of this file area. It begins with the first record and processes each record in sequence.

CONTROL-SEQUENTIAL-A and CONTROL-SEQUENTIAL-B

In the CONTROL-SEQUENTIAL access mode, records are stored with a blank (sequence-link) field appended to each record as it is loaded onto the disk pack. CONTROL-SEQUENTIAL-A refers to records without record marks;

CONTROL-SEQUENTIAL-B refers to records with record marks. Records to be added to the file are written into a separate area of disk storage. The address of the added record is written in the sequence-link field of the record that sequentially precedes the added record. Similarly, a record can be deleted and the sequence reestablished by placing the address of the following record in the sequence-link field of the preceding record. When the file is processed, the program checks the sequence-link of a record. If it is a blank, the next consecutive disk location is read. However, if the sequence-link field contains an address, the program seeks and reads the record stored at that address.

The length of the sequence-link field is a:

1. Six-digit address, if the file is unblocked, or a
2. Seven-digit address, if the file is blocked and the blocking factor is two to ten records per block, or a
3. Seven-digit address with zone bits over the seventh position, if the file is blocked and the blocking factor is eleven to thirty records per block, or an
4. Eight-digit address, where the eighth digit identifies the record mark (‡).

Standard six-digit addresses are used in an unblocked file. A seventh digit (R) is added to the addresses used with blocked files, when the blocking factor is two to ten records per block. These addresses are in the form SSSSSSR. The first six digits are the address of the first sector of the block. The seventh digit (R) designates the position of the record within the block. An eighth digit (R) is added for a record mark. These addresses are in the form SSSSSSRR. The first six digits are the address of the first sector of the block. The seventh digit designates the position of the record within the block. The eighth digit is used to identify a record mark.

The COBOL processor handles the various disk record forms as shown in Figure 22.

ACCESS MODE	FILE TYPE	RECORD FORM PERMITTED?		
		FIXED-LENGTH		VARIABLE-LENGTH
		UNBLOCKED (Form-1)	BLOCKED (Form-2)	BLOCKED (Form-4)
RANDOM	INPUT OR INPUT-OUTPUT	Yes	Yes	No
RANDOM	OUTPUT	Yes	No	No
CONTROL SEQUENTIAL	INPUT OR INPUT-OUTPUT	Yes	Yes	No
SEQUENTIAL	INPUT	Yes	Yes	Yes
SEQUENTIAL	OUTPUT	Yes	Yes	Yes

Figure 22. Record Formats for Disk Files

The maximum blocking factor (number of logical records per block) is ten for random files, one hundred for sequential files, and thirty for control-sequential files.

[ACTUAL KEY IS *data-name*]

This clause is required if the access mode of a mass-storage file has been specified as RANDOM.

Data-name is the name given by the programmer to the core-storage field that will contain the disk address of the record currently being processed in a given file during execution of the object program. This field will be updated in one of two ways:

1. By PROCEDURE DIVISION statements written by the source programmer.
2. By statements developed by the processor from specifications given in the KEY-CONVERSION section in association with the USE verb.

The format of the data-name field must be: SSSSSSR. The first six digits (SSSSSS) are the actual address of the disk sector where the block of records is stored. If the block of records covers more than one sector, the data-name field contains the address of the first sector. The seventh digit (R) indicates which record in the block is to be made available for processing. This digit (R) may be any digit 0-9. The digit 0 represents the first record in the block; the digit 1 represents the second record, etc. If the file consists of unblocked, random-access records, the seventh digit (R) must be 0.

[SYMBOLIC KEY IS *data-name*]

The SYMBOLIC KEY clause must be used only if a KEY-CONVERSION has been specified for a RANDOM file. If no KEY-CONVERSION has been specified, the SYMBOLIC KEY clause must be omitted.

Data-name is the field operated on in the section that makes the KEY-CONVERSION (see *Procedure Division*). The SYMBOLIC KEY is the indirect reference to data-name.

Example (Figure 23): To compute the address of any record needed by this program, the factor +3000 is added to the contents of the SYMBOLIC KEY field and the sum is divided by 25. The result is the ACTUAL KEY. (See *Declarative* section for a description of the USE verb.)

```
CONT
      A   B
  7   8  12  16  20  24  28  32  36  40  44  48
      CONVERT SELECTION.
          USE FOR KEY-CONVERSION ON INPUT-FILE.
      ADDRESS-ROUTINE. COMPUTE ACTUAL KEY =
          ( SYMBOLIC KEY + 3000 ) / 25.
```

Figure 23. KEY-CONVERSION

FILE-LIMIT[S] { IS / ARE } *integer* { THRU / THROUGH } *integer*

These two integers are the sector addresses of the first and last physical records of the file. Each integer contains six numeric characters.

If the RDLIN macro is used to redefine the file limits of the file, the new labels must not create the need for handling the cylinder-overflow condition if the file limits used at compile time did not imply cylinder overflow.

Example: The lower and upper limits for a given file are 000199 and 002199. Figure 24 shows a correct FILE-LIMITS entry.

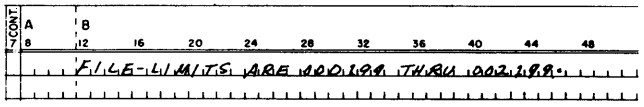


Figure 24. FILE-LIMITS

$$\left[\ \underline{\text{VALUE}}\ \underline{\text{OF}}\ \textit{data-name}\ \text{IS}\ \textit{literal}\ \left[\textit{data-name}\ \text{IS}\ \ldots\right]\right]$$

The VALUE OF clause is used in a file-description entry if the file has standard label records. The following sets of keywords and descriptions of literals are used as entries of this clause.

Data-Name	*Literal*
{ ID / IDENTIFICATION }	10-character AN
CREATION-DATE	5-digit numeric
RETENTION-CYCLE	4-digit numeric
FILE-SERIAL-NUMBER	5-digit numeric
FILE-SEQUENCE-NUMBER	4-digit numeric
PACK-SERIAL-NUMBER	5-digit numeric

Use of the PACK-SERIAL-NUMBER entry implies complete label checking for this file.

The processor selects particular label-checking procedures based upon the entries in the VALUE OF clause. The relations between the entries, the types of files, and the label-checking procedures selected are shown in Figure 25.

Disk Trailer Labels

The following information is contained in IBM standard disk trailer labels.

Trailer Label	*Positions*	*Contents*
(contains as many characters as the user's records)	1-5	1 EOR b 1 EOF b

Record-Description Entries

This section supplements the clause descriptions given in the COBOL General Information Manual.

CHECKING ROUTINES SELECTED	Complete Label Checking		Partial Label Checking
	INPUT	OUTPUT	INPUT
ID	10 AN	10 AN	10 AN
CREATION-DATE	5		
RETENTION-CYCLE	4	4	
FILE-SERIAL-NUMBER	*5	*5	
FILE-SEQUENCE-NUMBER	**4	**4	
PACK-SERIAL-NUMBER	5	5	

* If not present, the pack-serial-number will be used.
** If not present, 0001 will be assumed.

Figure 25. Label Checking

SIZE

This clause tells the processor how many characters (or digits) the data item contains. The general reference format for a SIZE clause is:

$$\left[\ \underline{\text{SIZE}}\ \text{IS}\ \left[\textit{integer-1}\ \text{TO}\right]\ \textit{integer-2}\ \left[\ \left\{\begin{matrix}\text{CHARACTER[S]}\\ \text{DIGIT[S]}\end{matrix}\right\}\ \right]\right]$$

$$\left[\ \underline{\text{DEPENDING}}\ \text{ON}\ \textit{data-name}\ \right]$$

This size is interpreted by the COBOL processor in terms of characters if either the optional word CHARACTER[S] or DIGIT[S] is used or if neither of the optional words is used.

To specify the sizes of variable-length records, (form-4) *integer-1* and *integer-2* and DEPENDING ON *data-name* must be used. *Integer-1* specifies the number of characters in the smallest record and *integer-2* specifies the number of characters in the largest record. DEPENDING ON *data-name* identifies the elementary items whose value is the record character count (refer to *Record Character-Count Field*). *Integer-1* and DEPENDING ON *data-name* may be used only with form-4 records.

Example: Figure 26 shows a SIZE entry for a form-4 record which can contain from 50 to 150 characters. RECCOUNT is the data-name the programmer has used to identify the RCC field.

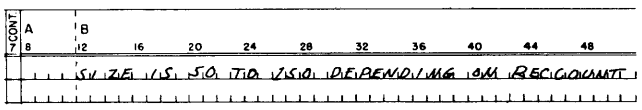


Figure 26. SIZE Variable Length

The size of fixed-length records is specified by using the form:

$$\left[\underline{\text{SIZE}}\ \text{IS}\ \textit{integer-2}\left[\begin{Bmatrix}\text{CHARACTER[S]}\\ \text{DIGIT[S]}\end{Bmatrix}\right]\right]$$

where *integer-2* is the exact number of characters contained in the record or item of data.

Example: Figure 27 shows a SIZE entry for a fixed-length record whose size is 80 characters.

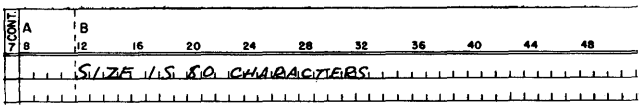


Figure 27. SIZE Fixed Length

VALUE

The general reference format for a VALUE clause is:

$$\left[\begin{Bmatrix}\underline{\text{VALUE}}\ \text{IS}\\ \underline{\text{VALUES}}\ \text{ARE}\end{Bmatrix}\ \textit{literal-1}\left[\underline{\text{THRU}}\ \textit{literal-2}\right]\right.$$

$$\left.\left[\textit{literal-3}\quad\left[\underline{\text{THRU}}\ \textit{literal-4}\right]\ldots\right]\right]$$

A VALUE clause can state the initial contents (VALUE) of a data item in the WORKING-STORAGE SECTION or CONSTANT SECTION. It can also be used to define the value of a condition-name (level-88 item) in the FILE SECTION and WORKING-STORAGE SECTION.

If the VALUE clause is not used to define the initial values of WORKING-STORAGE ITEMS, their contents at program execution time will be unpredictable.

If the VALUE clause specifies a numerical literal, an operational sign will be developed (placed over the units position of the numerical field), only if the literal is preceded by a plus or minus sign. A figurative constant may be used in the VALUE entry where a literal is specified.

The THRU option is not described in the COBOL General Information Manual. It may be used only with condition-names as shown in Figure 28.

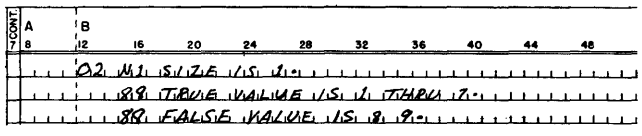


Figure 28. THRU Option

The Constant and Working Storage Sections

The record-description entries described for the FILE SECTION apply also to the CONSTANT and WORKING-STORAGE SECTIONS.

These sections begin with the header line WORKING-STORAGE SECTION or CONSTANT SECTION and are followed immediately by the record-description entries.

Added Elective Elements of the Data Division

These elective elements of the record-description entry are not specified in the COBOL General Information Manual, but are contained in this COBOL processor:

1. The DEPENDING ON *data-name* and the TO *integer-2* options of the SIZE clause.
2. The THRU *literal-2* and the *literal-3* THRU *literal-4* options of the VALUE clause.
3. All entries on mass storage.

Deferred Elements of the Data Division

1. The COPY option is contained in the COBOL General Information Manual, but is not contained in this COBOL processor.
2. The following editing functions cannot be specified by editing clauses or picture clauses:
   a. Editing of a single digit field.
   b. Single-position zero suppression. For example, Z9 is incorrect but ZZ is correct.
3. No item may exceed 999 characters.

Not Applicable

The USAGE, SIGNED, and SYNCHRONIZED clauses have no meaning in a 1401, 1440, or 1460 COBOL program and should not be used.

Note: A decimal insertion character cannot be used as the rightmost PICTURE character. For example, the following entry will not be considered valid:

02 NAME PICTURE Z,ZZZ,ZZ9. OCCURS 12 TIMES.

Procedure Division

The PROCEDURE DIVISION is the operational part of the COBOL source program. Once the data has been described, the programmer tells the COBOL processor what steps the machine must take to read the input data, process it, and write it as output on punched cards, magnetic tape, or a printed form.

The COBOL verbs are the main elements in the PROCEDURE DIVISION. They are described in detail in the COBOL General Information Manual. However, some verbs have special meaning when used in a 1401, 1440, and 1460 COBOL source program. This additional information is presented in the following section.

Declaratives

Reference Format.

DECLARATIVES.

Section-name-1 SECTION.

USE FOR KEY-CONVERSION ON

$$\left\{\begin{matrix}\underline{\text{ALL}}\ \text{FILES}\\ \textit{file-name}\left[\textit{file-name}\right]\ldots\end{matrix}\right\}.$$

Paragraph-name. Any procedure statement(s).

$$\left[\textit{Section-name}\ \underline{\text{SECTION}}.\ \underline{\text{USE}}\ldots\right]$$

END DECLARATIVES.

Declaratives are procedures that operate either under control of the main body of the PROCEDURE DIVISION or under control of the Input/Output Control System. They consist of sentences and associated procedures designed to give special information to the COBOL compiler.

If declaratives are used in a COBOL source program:

1. They must be grouped together and placed at the beginning of the PROCEDURE DIVISION, and
2. The group of declaratives must be preceded by the key word, DECLARATIVES, and must be followed by the keywords, END DECLARATIVES.

   Each declarative occupies a single section and must conform to the rules of procedure formation as described in the *Procedure Division* section of the COBOL General Information Manual. The source programmer must write the DECLARATIVES and END DECLARATIVES entries beginning in column 8.

The USE declarative is used in the 1401, 1440, and 1460 COBOL to specify the KEY-CONVERSION procedure which is to be used for developing disk addresses. This enables the source programmer to supply his own conversion factors and techniques for obtaining disk addresses.

A USE declarative may be used to specify the KEY-CONVERSION for more than one file. Thus, if a general key conversion algorithm must operate on different data names for different files, the ACTUAL KEY and SYMBOLIC KEY clauses may be used. These clauses appear in the MD entries of the DATA DIVISION. (See *Mass-Storage Files.*) Each MD that specifies the ACTUAL KEY and SYMBOLIC KEY clauses implies a USE declarative. The processor will associate the ACTUAL KEY and SYMBOLIC KEY functions by file.

Example: In the example shown in Figure 29, the same key-conversion procedure is used for two different files. The MD entries inform the processor of the particular data-names which must be associated with ACTUAL KEY and SYMBOLIC KEY for each file. When the disk addresses for file records are computed at object-program execution time, the contents of the SYMBOLIC KEY field will be added to FACTOR-1 and the sum will be divided by FACTOR-2. The result is the ACTUAL KEY.

The DISPLAY Verb

The printer is the standard output unit for the DISPLAY verb. However, information may also be displayed via the card read-punch or the console printer. As many printer lines or punched cards will be used as are necessary to display the information contained in the area of core storage whose data-name is specified in the DISPLAY statement.

The object program initiates a skip to channel 1 in the carriage tape if a form overflow occurs in the printer. If the DISPLAY verb is used in the PROCEDURE DIVISION to address the printer, the processor assumes that the printer will have a carriage tape with punches in channels 1 and 12 (overflow) at object program execution time.

Examples: The statement shown in Figure 30 will cause the contents of the area whose data-name is GRAND-TOTAL to be displayed on the printer.

The statement shown in Figure 31 will cause the contents of GRAND-TOTAL to be punched into cards, if the mnemonic-name CARD-PUNCH has been assigned to 1402-P or 1442-P or 1444-P in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

The ACCEPT Verb

A card reader is the standard input device for the ACCEPT verb. However, the console printer can also serve as an input device. When ACCEPT is from a card reader, the minimum area that can be declared is 80 positions.

Example: Figure 32 shows an ACCEPT statement that will cause data to be read from the card reader and moved to an area whose data-name is CANCELLATIONS. If more than 80 storage positions are defined by CANCELLATIONS, multiple cards will be read from the card reader until the area is filled.

The ENTER Verb

The ENTER verb permits the programmer to use Autocoder statements in a COBOL source program.

The *language-name* used with 1401, 1440, and 1460 COBOL is AUTOCODER. The Autocoder statements must be presented to the COBOL processor immediately following the ENTER AUTOCODER statement, and they must be followed by an ENTER COBOL entry that indicates the point at which the COBOL source language is resumed. Each ENTER AUTOCODER statement must constitute a separate paragraph in the source program and must appear on the same line as the name of the paragraph. The ENTER COBOL statement used for returning to COBOL from Autocoder must either constitute a separate paragraph or be the first entry of a paragraph. The name of this paragraph must be on the same line as the ENTER COBOL statement.

These specifications must be maintained when using Autocoder entries in a COBOL program:

1. Autocoder statements must be coded in Autocoder format (label starting in column 6, operation in column 16, and operand in column 21).
2. Symbols used in Autocoder statements must be five characters in length.
3. Macro instructions are permitted.
4. Autocoder statements can be written to refer to COBOL-names if they are related by entries in the

```
MD  FILE-1
        ACTUAL KEY IS DISK-ADDRESS-1
        SYMBOLIC KEY IS PART-NUMBER-1.
       .   .   .
MD  FILE-2
        ACTUAL KEY IS DISK-ADDRESS-2
        SYMBOLIC KEY IS PART-NUMBER-2.
       .   .   .
PROCEDURE DIVISION.
DECLARATIVES.
CONVERT-TO-DISK-ADDRESS SECTION.
        USE FOR KEY-CONVERSION ON ALL FILES.
ADDRESS-ROUTINE. COMPUTE ACTUAL KEY = ( SYMBOLIC KEY + FACTOR
-       -1 ) / FACTOR-2.
END DECLARATIVES.
IMPERATIVE SECTION.
START. OPEN I-O FILE-1 FILE-2.
        READ FILE-1 INVALID GO TO FAILURE-1 IF X=Y PERFORM WRITE-BACK
-       -1.
        READ FILE-2 INVALID GO TO FAILURE-2 IF A=B PERFORM WRITE-BACK
-       -2.
        GO TO START.
```

Figure 29. KEY-CONVERSION Sample (Part 1 of 2)

```
FAILURE-1.
       .   .   .
       .   .   .
       .   .   .
FAILURE-2.
       .   .   .
       .   .   .
       .   .   .
WRITE-BACK-1.
       .   .   .
       .   .   .
       .   .   .
WRITE-BACK-2.
       .   .   .
       .   .   .
       .   .   .
```

Figure 29. KEY-CONVERSION Sample (Part 2 of 2)

```
        DISPLAY GRAND-TOTAL
```

Figure 30. Printer DISPLAY

```
        DISPLAY GRAND-TOTAL UPON CARD-PUNCH
```

Figure 31. Punch DISPLAY

```
        ACCEPT CANCELLATIONS
```

Figure 32. ACCEPT

SPECIAL-NAMES section of the COBOL program. However, COBOL statements cannot be written to refer to Autocoder-names.

5. The word-mark status of a constant or area defined by a COBOL statement must be the same after the Autocoder statements are executed in the object program as it was before they were executed. Thus, if it is necessary to write an Autocoder statement that sets or clears a word mark in such an area, the word-mark position of that area must be tested first so that the word mark can be reset or cleared before returning to the COBOL program.
6. No 1401 SPS statements can be included.
7. When executing overlays while in the ENTER AUTOCODER mode, use of the COBOL OVLAY macro causes the correct branch-to-the-loader to be generated.

Example: Figure 33 is an example that includes a section of Autocoder statements.

The OPEN Verb

Reference Format.

$$\underline{\text{OPEN}} \left\{ \begin{array}{l} \underline{\text{INPUT}} \\ \underline{\text{OUTPUT}} \\ \left\{ \begin{array}{l} \underline{\text{INPUT-OUTPUT}} \\ \underline{\text{I-O}} \end{array} \right\} \end{array} \right\} \textit{file-name-1} \left[\left[\textit{file-name-2} \ldots \right] \left\{ \begin{array}{l} \underline{\text{INPUT}} \\ \underline{\text{OUTPUT}} \\ \left\{ \begin{array}{l} \underline{\text{INPUT-OUTPUT}} \\ \underline{\text{I-O}} \end{array} \right\} \end{array} \right\} \textit{file-name-n} \ldots \right]$$

The set of rules shown in Figure 34 applies to the file types specified in OPEN statements:

*Specifying a file as an INPUT-OUTPUT file assumes that:

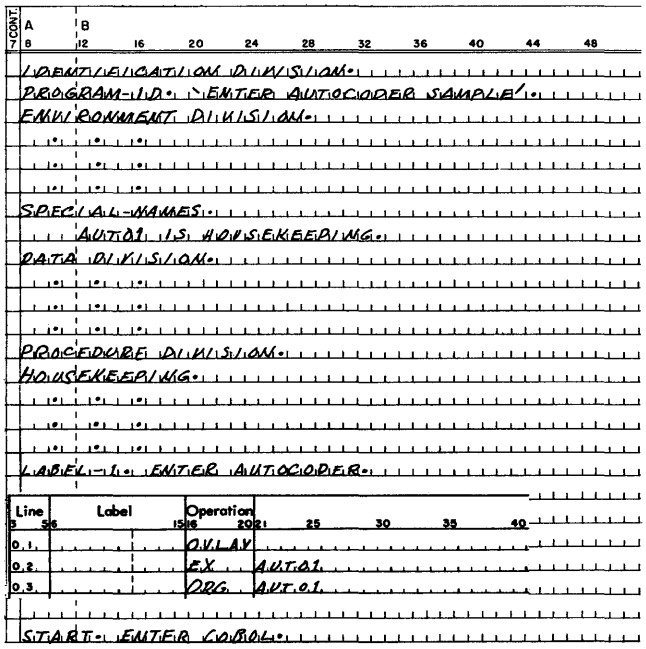


Figure 33. ENTER Sample

1. The file must have been described in an MD entry.
2. Any use of the WRITE verb in association with this file will cause the specified record to be written back on the file in the position referred to by the last READ associated with this file.

FILE TYPE	RANDOM	CONTROL-SEQUENTIAL	SEQUENTIAL
INPUT	YES	YES	YES
OUTPUT	YES	NO	YES
INPUT-OUTPUT	* YES	* YES	NO

Figure 34. File Types for OPEN Statements

Note: A file specified as having a CONTROL-SEQUENTIAL operation mode may not be opened as an OUTPUT file.

The READ Verb

Reference Format for mass-storage random INPUT or random INPUT-OUTPUT files:

READ *file-name* RECORD [INTO *data-name*]
[INVALID KEY *any-imperative-statement*]

Reference Format for all other INPUT or INPUT-OUTPUT files:

READ *file-name* RECORD [INTO *data-name*]
[AT END *any-imperative-statement*]

This statement causes a logical record to be released from an INPUT or INPUT-OUTPUT file and transferred to the record-name associated with the file.

Data-name is the name given by the programmer to the core-storage area to which the record must be transferred. After the READ statement is executed, the logical record will be available both in *record-name* and in *data-name.*

Both the INVALID KEY and the AT END options may be implied within the COBOL program. This means that the appropriate clause must be used at least once in the COBOL program in association with each INPUT or INPUT-OUTPUT file. If a given INPUT or INPUT-OUTPUT file has *only one* use of the INVALID KEY or AT END option, all READ statements associated with the file will assume the implied option. If more than one option is used with a file, it is required that all READ statements have explicit INVALID KEY or AT END options.

The *any-imperative-statement* is executed as described here:

Nature of File	*Appropriate Options*	*Triggering Condition*
Card Reader	AT END	Attempt to read when hopper is empty.
Random File	INVALID KEY	The ACTUAL KEY is either outside the limits defined in the FILE-LIMITS clause or is an invalid disk address.
All Others	AT END	When an end-of-file condition or the upper file limit is encountered.

The SEEK Verb

Reference Format.

SEEK *file-name* RECORD

This verb allows the user to seek a particular record as specified in the ACTUAL KEY statement. Processing continues while the seek operation is performed until the next READ or WRITE disk file statement is encountered. If the programmer has not specified a SEEK *file-name* RECORD, a seek instruction will be automatically executed in the object program before the disk read or write operation.

The STOP Verb

Reference Format.

STOP { *literal* / RUN }

This statement produces a machine HALT instruction which stops the execution of the object program. The RUN option of the STOP verb causes an unconditional halt, and the program cannot be restarted.

If the STOP *literal* option is used, and if the object computer has a console printer, the literal will be displayed upon it. Otherwise, if the stop literal is numeric and within the range 00-99, the literal itself is displayed in the B-register if the halt occurs during the running of the object program.

If the stop literal is numeric and greater than 99 or if it is alphanumeric, the address of the literal is displayed in the B-address register if an object program halt occurs. Pressing the start key allows the object program to proceed.

The WRITE Verb

Reference Format for control-sequential files and punch file:

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

Reference Format for printer file:

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

[{BEFORE / AFTER} ADVANCING {*integer* LINES / *mnemonic-name*}]

Reference Format for sequential files and random files:

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

[INVALID KEY *any-imperative statement*]

This statement causes a logical record to be released for an output file.

Record-name is the name given to the record defined at the 01 level in the FILE SECTION of the DATA DIVISION under the FD or MD entry for the associated file-name. *Area-name* is the name given by the programmer to the core-storage area from which the record is to be written.

The ADVANCING option is used for spacing lines on output documents on the printer. The number of spaces skipped is equal to the normal printer skip plus the integer specified in the ADVANCING option. The integer specified will be treated as meaning the number of spaces between lines of print.

AFTER and BEFORE in the ADVANCING option control printer carriage spacing or skipping before or after the WRITE verb is executed. *Integer* LINES specifies how many lines should be spaced. *Mnemonic-name* is the name assigned in the SPECIAL-NAMES paragraph to a channel in the carriage tape and is used when carriage skipping is desired instead of line spacing. The skip occurs to the line that corresponds to the specified punch in the carriage tape.

The INVALID KEY option of the WRITE verb, used only with SEQUENTIAL or RANDOM mass-storage files, permits the source programmer to specify the appropriate action to be taken when IOCS senses the upper limit specified in the FILE-LIMITS entry. This option follows the same set of rules as the INVALID KEY or AT END option of the READ verb in connection with implied statements.

Note: The INVALID KEY option should not be used with the WRITE verb for INPUT-OUTPUT files. The INVALID KEY option of the READ verb will serve as the test for these files.

Examples: Figures 35, 36, 37, and 38 show sample WRITE statements.

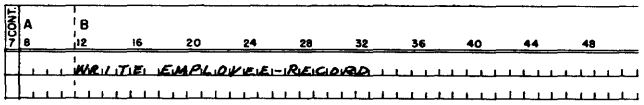


Figure 35. WRITE

Exponents

IBM 1401, 1440, and 1460 COBOL provides for integer or non-integer powers to be used in exponentiation. The sign of the power can be either plus or minus. Please note that negative bases cannot be raised to other than an integer power.

Conditional Statements

Option 1

IF *conditional expression statement-1.*

Option 2

IF *conditional expression* {*statement-2* / NEXT SENTENCE} {OTHERWISE / ELSE} {*statement-3* / NEXT SENTENCE}

Option 3

{{*statement-10* INVALID KEY / *statement-4* AT END / *statement-5* ON SIZE ERROR} {*statement-6* / NEXT SENTENCE} [{OTHERWISE / ELSE} {*statement-7* / NEXT SENTENCE}] / any imperative *statement-8* followed by any conditional *statement-9*}

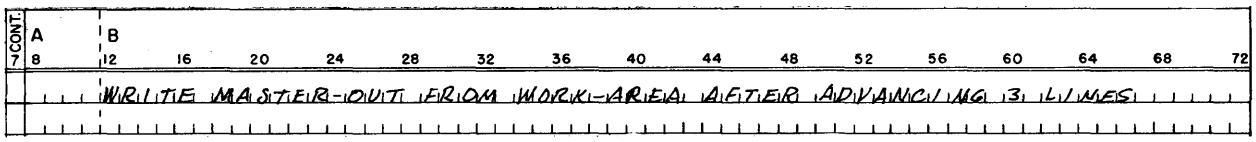


Figure 36. WRITE AND SPACE BEFORE PRINTING

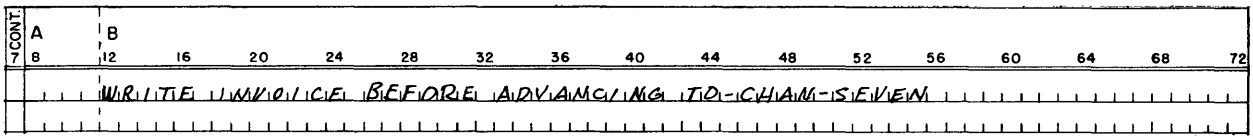


Figure 37. WRITE AND SKIP AFTER PRINTING

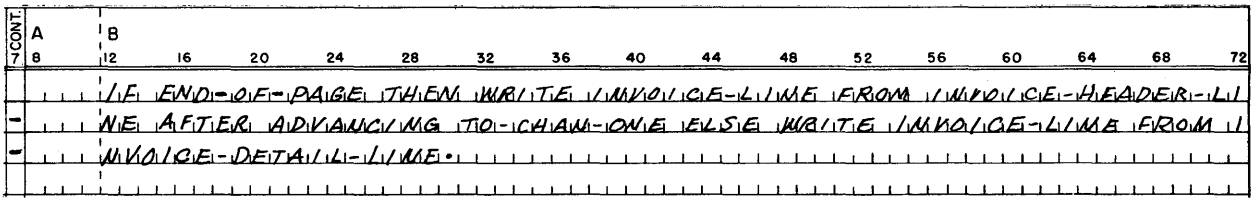


Figure 38. CONDITIONAL WRITE

Statement-1 under Option 1 can be only a simple or compound imperative statement.

Statement-2 and/or *statement-3* under Option 2 and *statement-7* under Option 3 can be either imperative or conditional. If conditional, these statements can contain conditional statements in arbitrary depth. When conditional, the conditions within the conditional statements are nested.

Statement-4 under Option 3 must be a READ statement, *statement-5* must be an arithmetic statement, and *statement-6* can be only a simple or compound imperative statement. *Statement-8* followed by *statement-9* (to which the previous paragraph applies because it is conditional) is an illustration of an imperative statement followed by a conditional statement. This is logically equivalent to *statement-8* followed by a period followed by *statement-9* beginning a new sentence. Option 3 in its entirety may be substituted for *statement-2* and/or *statement-3* under Option 2.

An ELSE or OTHERWISE must be explicitly written for every conditional statement within a sentence. However, the phrase ELSE (OTHERWISE) NEXT SENTENCE may be eliminated only if the phrase immediately precedes the period ending a sentence.

Statement 10 must be either a READ statement (associated with a disk file or a WRITE statement (associated with a disk file).

Nested Conditional IF Statements

The COBOL programmer can combine several simple conditional statements into one by using a technique called *nesting*. The processor analyzes a nested statement by working from the inside to the outside of the statement. Thus, if all conditions are satisfied, the first imperative is executed; if all but the last condition are satisfied, the second imperative is executed, etc.

Figure 39 shows outlines for four simple conditional statements. Figure 40 shows an outline for one nested conditional IF statement that produces the same results as the four simple conditional statements shown in Figure 39.

Figure 41 shows an excerpt from a COBOL program in which four simple relational conditional expressions are substituted for the conditions shown in Figures 39 and 40.

The block diagram in Figure 42 shows the logic flow of the nested IF statement in Figure 41.

Added Elective Elements of the Procedure Division

The following options are not contained in the COBOL General Information Manual but are contained in this COBOL processor.

1. The ADVANCING option of the WRITE verb.
2. Option 3 of conditional statements.
3. The nesting of conditional statements.

Deferred Elements of the Procedure Division

These features described in the COBOL General Information Manual are not implemented by this COBOL processor.

1. The entire COPY option.
2. The CORRESPONDING option of the MOVE verb.
3. The REEL option of the CLOSE verb.

Notes:

1. A COBOL source program can be compiled that produces as many as 4,000 Autocoder and IOCS statements which, when expanded, may produce as many as 6,000 one-for-one Autocoder statements.
2. In order to ensure correct decimal alignment when using the DIVIDE verb with the GIVING option, the programmer must declare a result field, the decimal portion of which is no more than one position greater than the decimal portion of the dividend.
3. When NOTE is the first word of a paragraph, it must appear on the same line as the paragraph name.
4. The figurative constant ZERO (ZEROS or ZEROES) cannot be used in an arithmetic computation. For example the statement COMPUTE *data-name* = ZERO is not allowed. This statement must be in the form of COMPUTE *data-name* = 0 .

```
IF (condition 1) AND (condition 2) AND (condition 3) AND (condition 4) GO TO LAB4 ELSE NEXT SENTENCE.
IF (condition 1) AND (condition 2) AND (condition 3) GO TO LAB3 ELSE NEXT SENTENCE.
IF (condition 1) AND (condition 2) GO TO LAB2 ELSE NEXT SENTENCE.
IF (condition 1) GO TO LAB1 ELSE NEXT SENTENCE.
```

Figure 39. Four Conditional IF Statements

```
IF (condition 1) IF (condition 2) IF (condition 3) IF (condition 4) GO TO
LAB4 ELSE GO TO LAB3 ELSE GO TO LAB2 ELSE GO TO LAB1 ELSE NEXT SENTENCE.
```

Figure 40. Nested Conditional IF Statements

```
CONT  A   B
 7    8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
      START. IF A = B IF C = D IF E = F IF G = H GO TO LABEL-4 ELSE GO
             TO LABEL-3 ELSE GO TO LABEL-2 ELSE GO TO LABEL-1 ELSE NEXT
             SENTENCE.

      LB. MOVE '4' TO ANSWER. GO TO START.

      LABEL-1. MOVE Z TO ANSWER. GO TO START.

      LABEL-2. MOVE W TO ANSWER. GO TO START.

      LABEL-3. ADD '2' TO ANSWER. GO TO START.

      LABEL-4. SUBTRACT Z FROM W GIVING ANSWER. GO TO START.
```

Figure 41. Program Sample for Nested Conditional IF Statements

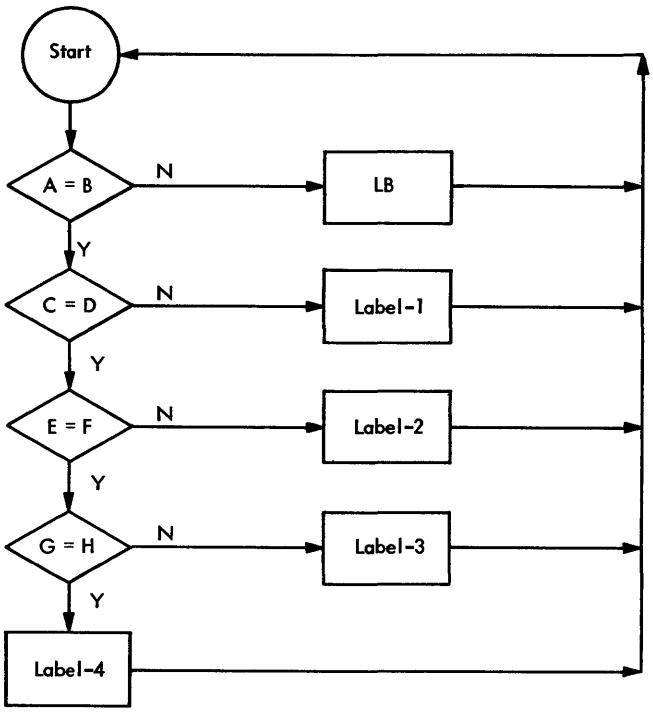


Figure 42. Conditional Logic

Character Sets

IBM Character Set H must be used for source programs. This character set consists of the numerals 0 through 9, the 26 letters of the alphabet, and 12 special characters. The machine character set may be used only for alphanumeric literals. The following are COBOL (Set H) special characters with their equivalents in the IBM 1401, 1440, and 1460 character set:

Card Code	COBOL *(Set H)*	*1401, 1440, 1460*	*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

LOW-VALUE(S)

The value of this figurative constant is the space, or blank. The blank character is the lowest in the IBM collating sequence.

HIGH-VALUE(S)

This figurative constant is defined as the integer 9. The character 9 is the highest in the IBM collating sequence.

QUOTE(S)

This figurative constant is defined as the COBOL character (Set H) for the quotation mark.

Word Lists

Additional COBOL Words

The following words constitute an extension of the list of COBOL words contained in the IBM General Information Manual describing COBOL. ID may be used in place of IDENTIFICATION. The meaning and use of the other words have been described in this bulletin.

1301-D	IBM-1401
1311-D	IBM-1440
1401-SS	IBM-1460
1402-P	I-O
1402-R	ID
1403-CT	INVALID
1403-P	KEY
1403-P-C9	KEY-CONVERSION
1403-P-CB	LABEL-INFORMATION
1403-P-CV	LINES
1440-SS	MD
1442-P	MODE
1442-R	NO-CONSOLE-PRINTER
1443-CT	NO-DIRECT-SEEK
1443-P	NO-MULTIPLY-DIVIDE
1443-P-C9	NO-OVERLAP
1443-P-CB	NO-PRINT-STORAGE
1443-P-CV	PACK-SERIAL-NUMBER
1444-P	PROCESSING
1447-CP	RANDOM
1460-SS	REEL-SEQUENCE-NUMBER
ACCESS	RETENTION-CYCLE
ACTUAL	SECTOR
ADVANCING	SEEK
BEFORE	SEQUENTIAL
CONTROL-SEQUENTIAL-A	SYMBOLIC
CONTROL-SEQUENTIAL-B	TAPE
CREATION-DATE	TAPES
DECLARATIVES	TYPE-A-LABEL
FILE-LIMIT(S)	TYPE-B-LABEL
FILE(S)	TYPE-C-LABEL
FILE-SERIAL-NUMBER	USE
FILE-SEQUENCE-NUMBER	VALUES

Class Conditions

The general information manual specifies that the *class* of a data item is either numeric, alphabetic or alphanumeric. It further specifies that the *class condition* tests an ALPHANUMERIC item at object time to determine whether it is wholly numeric or wholly alphanumeric 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*.

```
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 processor, depending upon which of the *class* tests is specified. The table shows the result (YES or NO) for each test and for each of the specified ranges of "X." The X-character is used in the PICTURE clause. It represents any character in the 1401, 1440, or 1460 character set.

x-Character	*If Numeric*	*If Alphabetic*
0-9	Yes	No
SPECIAL CHARACTERS	No	No
SPACE	No	Yes
A-R	Yes (if units position)	Yes
S-Z	No	Yes

Continuation of Alpha Literals

Alphanumeric literals must be preceded and followed by quotation marks. If an alphanumeric literal must be continued, a continuation (-) must appear in column 7 and a quotation mark must precede the remaining position of the literal. The quotation mark must be in the appropriate column for the particular division in which the literal appears. If the last character of an alphanumeric literal appears in column 72, column 7 of the next line must contain a continuation symbol and the next two significant characters in that line must both be quotation marks.

Reference Formats

Here is a summary of the reference formats used in writing a COBOL program for the IBM 1401, 1440, and 1460 Data Processing Systems with disk storage.

```
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.]
```

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. { IBM-1401 | IBM-1440 | IBM-1460 }
OBJECT-COMPUTER. { IBM-1401 | IBM-1440 | IBM-1460 }
[ MEMORY SIZE { { 4000 | 8000 | 12000 | 16000 } CHARACTERS
              | ADDRESS integer { THROUGH | THRU } { 4000 | 8000 | 12000 | 16000 } } ]
[ NO-PRINT-STORAGE ]
[ NO-MULTIPLY-DIVIDE ]
[ NO-DIRECT-SEEK ]
[ NO-OVERLAP ]
[ NO-CONSOLE-PRINTER ].
```

```
SPECIAL-NAMES.
[ device-name IS mnemonic-name
      [ device-name IS mnemonic-name . . . ] ].
[ switch-name { ON | OFF } STATUS IS condition-name
      [ { OFF | ON } STATUS IS condition-name ]
      [ switch-name . . . ] ].
[ autocoder-name IS cobol-name
      [ autocoder-name . . . . . ] ].
```

```
INPUT-OUTPUT SECTION.

FILE-CONTROL.  SELECT file-name 1

  ASSIGN TO device-name

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

[SELECT.....].

I-O-CONTROL.

APPLY {TYPE-A-LABEL / TYPE-B-LABEL / TYPE-C-LABEL} ON file-name [APPLY...].

DATA DIVISION.
FILE SECTION.
```

Tape Files:

```
FD file-name  [RECORDING MODE IS 1]

  [BLOCK CONTAINS integer-1  {RECORD[S] / CHARACTER[S]}]

  [RECORD CONTAINS  [integer-2 TO]

                        integer-3 CHARACTER[S]]

    LABEL RECORD[S] {IS / ARE} {STANDARD / OMITTED}

  [VALUE OF data-name-1 IS literal  [data-name-2 IS...]]

  DATA RECORD[S] {IS / ARE} data-name-3 [data-name-4...].
```

Punched-Card Files:

```
FD file-name [RECORDING MODE IS 1]

  [BLOCK CONTAINS integer-1 {RECORD[S] / CHARACTER[S]}]

  [RECORD CONTAINS  integer-3 CHARACTER[S]]

    LABEL RECORD[S]  {IS / ARE} OMITTED

    DATA RECORD[S]  {IS / ARE} data-name-3

        [data-name-4...].
```

Disk Files (Random Access):

```
MD file-name
[RECORDING MODE IS SECTOR ]
  PROCESSING MODE IS SEQUENTIAL
  ACCESS MODE IS RANDOM
  ACTUAL KEY IS data-name
[ SYMBOLIC KEY IS data-name ]
[ FILE-LIMIT[S]  {IS / ARE} integer {THRU / THROUGH} integer ]
[ BLOCK CONTAINS integer  {RECORD[S] / CHARACTER[S]}]
[ RECORD CONTAINS integer CHARACTERS ]
  LABEL RECORD[S] {IS / ARE} {STANDARD / OMITTED}
[ VALUE OF data-name IS literal [ data-name IS...]]
  DATA RECORD[S]  {IS / ARE} data-name [ data-name... ].
```

Control-Sequential Access:

MD *file-name*

[RECORDING MODE IS SECTOR]

PROCESSING MODE IS SEQUENTIAL

ACCESS MODE IS { CONTROL-SEQUENTIAL-A / CONTROL-SEQUENTIAL-B }

FILE-LIMIT[S] { IS / ARE } *integer* { THRU / THROUGH } *integer*

[BLOCK CONTAINS *integer* { RECORD[S] / CHARACTER[S] }]

[RECORD CONTAINS *integer* CHARACTER[S]]

LABEL RECORD[S] { IS / ARE } { STANDARD / OMITTED }

[VALUE OF *data-name* IS *literal* [*data-name* IS . . .]]

DATA RECORD[S] { IS / ARE } *data-name* [*data-name* . . .].

Sequential Access:

MD *file-name*

[RECORDING MODE IS SECTOR]

PROCESSING MODE IS SEQUENTIAL

ACCESS MODE IS SEQUENTIAL

FILE-LIMIT[S] { IS / ARE } *integer* { THRU / THROUGH } *integer*

[BLOCK CONTAINS *integer* { RECORD[S] / CHARACTER[S] }]

[RECORD CONTAINS [*integer* TO] *integer* CHARACTER[S]]

LABEL RECORD[S] { IS / ARE } { STANDARD / OMITTED }

[VALUE OF *data-name* IS *literal* [*data-name* IS . . .]]

DATA RECORD[S] { IS / ARE } *data-name* [*data-name* . . .]

Record Description:

level-number { FILLER / *data-name-1* } [REDEFINES *data-name-2*]

† [SIZE IS [*integer-1* TO] *integer-2* [{ CHARACTER[S] / DIGIT[S] }]] [DEPENDING ON *data-name*]

[OCCURS *integer-3* TIME[S]]

[POINT LOCATION IS { LEFT / RIGHT } *integer-4* PLACE[S]]

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

[PICTURE IS *any allowable combination of characters and symbols*]

[JUSTIFIED { LEFT / RIGHT }]

[{ ZERO SUPPRESS / CHECK PROTECT / FLOAT DOLLAR SIGN } [LEAVING *integer-5* PLACE[S]]]

[BLANK WHEN ZERO]

▲ † [{ VALUE IS / VALUES ARE } *literal-1* [THRU *literal-2*] [*literal-3* [THRU *literal-4*] . . .]]

* [USAGE IS { COMPUTATIONAL / DISPLAY }]

* [SIGNED]

* [SYNCHRONIZED { LEFT / RIGHT }]

*These clauses are not meaningful to this processor. If used, they will be ignored.

†These clauses have been designated as part of *Elective Cobol- 1961* and are not included in the General Information Manual.

▲ This clause is invalid if used in the FILE SECTION of the DATA DIVISION on other than 88 levels.

PROCEDURE DIVISION.

Option 1:

[*Section-name* SECTION.]

Paragraph-name. Any procedure statement(s).

Option 2:

DECLARATIVES.

Section-name-1 SECTION.

USE FOR KEY-CONVERSION ON { ALL FILES / *file-name*[*file-name*]. . . }

Paragraph-name. Any procedure statement(s).

[*Section-name* SECTION. USE . . .]

END DECLARATIVES.

Section-name-2 SECTION.

Paragraph-name. Any procedure statements).

ACCEPT *data-name* [FROM *mnemonic-name*]

ADD { *data-name-1* / *literal-1* } [{ *data-name-2* / *literal-2* } ...] [{ TO / GIVING } *data-name-n*]

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

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

CLOSE *file-name-1* [WITH { LOCK / NO REWIND }] [*file-name-2* ...]

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

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

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

procedure-name. ENTER AUTOCODER

procedure-name. ENTER COBOL

EXAMINE *data-name* (TALLYING { ALL / LEADING / UNTIL FIRST } *literal-1* [REPLACING BY *literal-2*] | REPLACING { ALL / LEADING / [UNTIL] FIRST } *literal-3* BY *literal-4*)

procedure-name. EXIT.

Option 1:

GO TO *procedure-name.*

Option 2:

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

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

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

NOTE *any comment.*

OPEN { INPUT / OUTPUT / { INPUT-OUTPUT / I-O } } *file-name-1* [[*file-name-2* ...] { INPUT / OUTPUT / { INPUT-OUTPUT / I-O } } *file-name-n*]

Option 1:

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

Option 2:

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

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* / *data-name-2* }

BY { *numeric-literal-2* / *data-name-3* } UNTIL *condition-1*

Option 5:

PERFORM *procedure-name-1* [THRU *procedure-name-2*] VARYING *subscript-name-1* FROM { *integer-1* / *data-name-1* }

BY { *integer-2* / *data-name-2* } UNTIL *condition-1* [AFTER *subscript-name-2* FROM { *integer-3* / *data-name-3* } BY { *integer-4* / *data-name-4* }

UNTIL *condition-2*] [AFTER *subscript-name-3* FROM { *integer-5* / *data-name-5* } BY { *integer-6* / *data-name-6* } UNTIL *condition-3*]

READ *file-name* RECORD [INTO *area-name*] [{ AT END / INVALID KEY } *any imperative statement*]

SEEK *file-name* RECORD

SUBTRACT { *data-name-1* / *literal-1* } [{ *data-name-2* / *literal-2* } ...] FROM { *data-name-n* / *literal-n* } [GIVING *data-name-n*] [ROUNDED]

[ON SIZE ERROR *any imperative statement*]

STOP { *literal* / RUN }

Option 1:

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

Option 2:†

WRITE *record-name* [FROM *area-name*] { AFTER / BEFORE } ADVANCING { *integer* LINES / *mnemonic name* }

Option 3:

WRITE *record-name* [FROM *area-name*] INVALID KEY *any-imperative-statement*

†This verb option has been designated as part of *Elective Cobol- 1961* and is not included in the General Information Manual.

1. Simple Relational Conditions *

```
[[{data-name          }]{ IS [NOT] GREATER THAN }      {data-name          }
 [{literal            }]{ IS [NOT] LESS THAN    }      {literal            }
 [{arithmetic expression}]{ IS [NOT] EQUAL TO   }      {arithmetic expression}
                          {  =                  }
```

2. Sign Conditions

```
{arithmetic expression} IS [NOT] {POSITIVE}
{data-name            }          {NEGATIVE}
                                 {ZERO    }
```

3. Class Conditions

```
data-name IS [NOT] {NUMERIC   }
                   {ALPHABETIC}
```

4. Condition-Names

```
[NOT] condition-name
```

5. Switch-Status-Names

```
[NOT] switch-status-name
```

Option 1:

```
IF conditional expression statement-1.
```

Option 2:

```
IF conditional expression {statement-2 }  {OTHERWISE}  {statement-3 }
                          {NEXT SENTENCE}  {ELSE     }  {NEXT SENTENCE}
```

Option 3:†

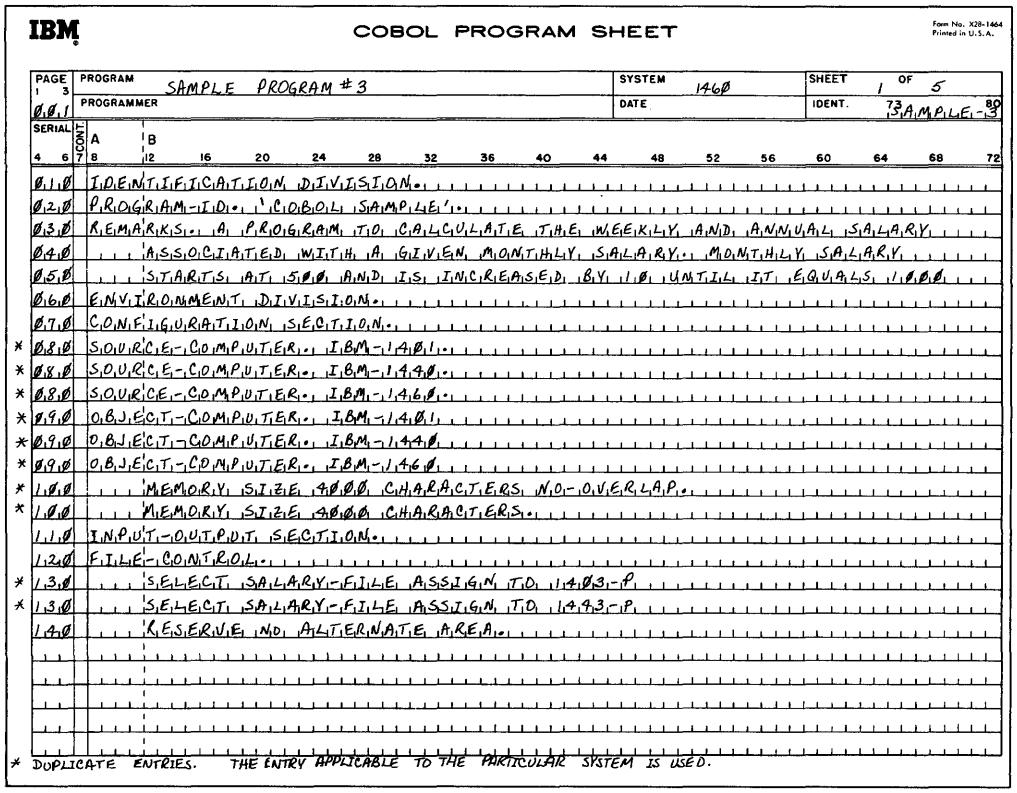
```
{{statement-10 INVALID KEY }{statement-6 }  {OTHERWISE}  {statement-7 }}
{{statement-4 AT END       }{NEXT SENTENCE}  {ELSE     }  {NEXT SENTENCE}}
{{statement-5 ON SIZE ERROR}                                            }
{
{  any imperative statement-8 followed by any conditional
{  statement-9
```

* These entries are optional only under the rules of implication described in the COBOL General Information manual. They require a complete, simple relational expression before any implications are used later in the same conditional statement.

† This conditional statement form has been designated as part of *Elective Cobol-1961* and does not appear in the General Information Manual.

Sample Problem

In this program, the calculation of the weekly and annual salary associated with a given monthly salary is coded in the COBOL language. The monthly salary starts at $500 and is increased by $10 until it equals $1,000 (Figure 43).



IBM COBOL PROGRAM SHEET

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

PAGE 1 3 | PROGRAM SAMPLE PROGRAM #3 | SYSTEM 1460 | SHEET 1 OF 5

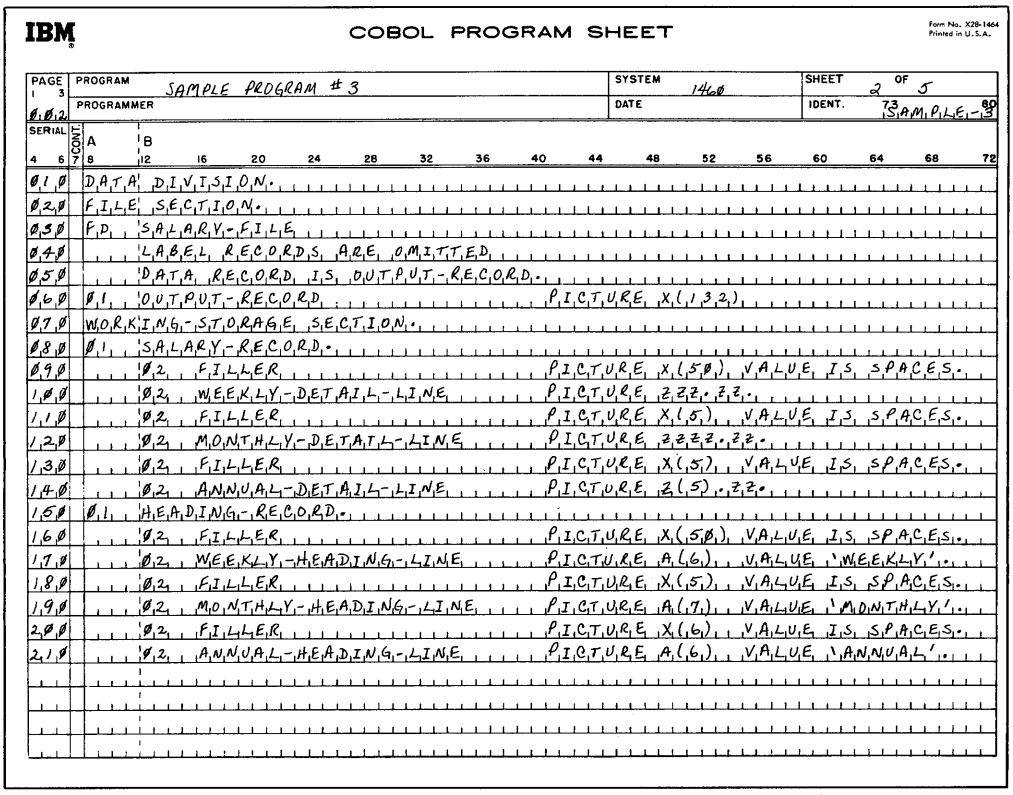
PROGRAMMER | DATE | IDENT. 73 SAMPLE-3 80

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

```
0010 IDENTIFICATION DIVISION.
0020 PROGRAM-ID. 'CBLSAMP3'.
0030 REMARKS. A PROGRAM TO CALCULATE THE WEEKLY AND ANNUAL SALARY
0040          ASSOCIATED WITH A GIVEN MONTHLY SALARY. MONTHLY SALARY
0050          STARTS AT $500 AND IS INCREASED BY 10 UNTIL IT EQUALS 1000
0060 ENVIRONMENT DIVISION.
0070 CONFIGURATION SECTION.
0080 SOURCE-COMPUTER. IBM-1401.
*0080 SOURCE-COMPUTER. IBM-1410.
*0080 SOURCE-COMPUTER. IBM-1460.
*0090 OBJECT-COMPUTER. IBM-1401.
*0090 OBJECT-COMPUTER. IBM-1410.
*0090 OBJECT-COMPUTER. IBM-1460.
*0100          MEMORY SIZE 4000 CHARACTERS NO-OVERLAP.
*0100          MEMORY SIZE 40000 CHARACTERS.
0110 INPUT-OUTPUT SECTION.
0120 FILE-CONTROL.
*0130          SELECT SALARY-FILE ASSIGN TO 1403-P
*0130          SELECT SALARY-FILE ASSIGN TO 1443-P
0140          RESERVE NO ALTERNATE AREA.
```

* DUPLICATE ENTRIES. THE ENTRY APPLICABLE TO THE PARTICULAR SYSTEM IS USED.

Figure 43. Sample Program #3 (part 1 of 5)



IBM

COBOL PROGRAM SHEET

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

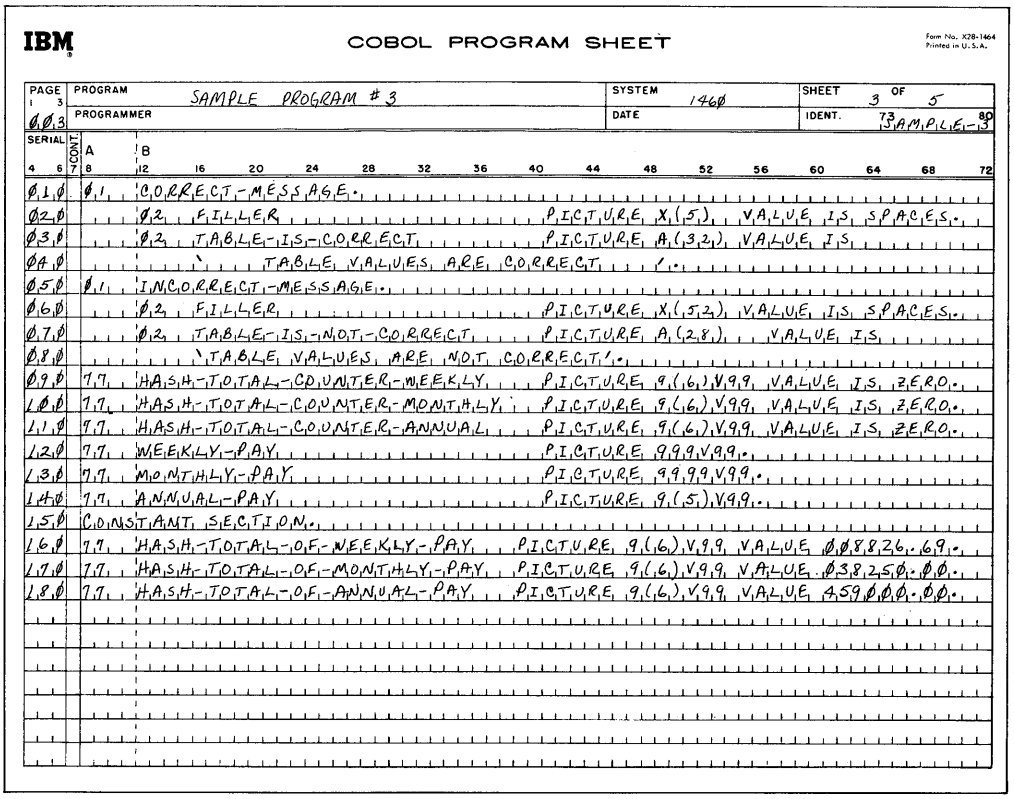
PAGE 1 3 | PROGRAM: SAMPLE PROGRAM #3 | SYSTEM: 1460 | SHEET 2 OF 5

PROGRAMMER | DATE | IDENT. 73 SAMPLE-3 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

```
01 0  DATA DIVISION.
02 0  FILE SECTION.
03 0  FD  SALARY-FILE
04 0      LABEL RECORDS ARE OMITTED
05 0      DATA RECORD IS OUTPUT-RECORD.
06 0  01  OUTPUT-RECORD                    PICTURE X(1322)
07 0  WORKING-STORAGE SECTION.
08 0  01  SALARY-RECORD.
09 0      02  FILLER                      PICTURE X(50)  VALUE IS SPACES.
10 0      02  WEEKLY-DETAIL-LINE          PICTURE ZZZ.ZZ.
11 0      02  FILLER                      PICTURE X(5)   VALUE IS SPACES.
12 0      02  MONTHLY-DETAIL-LINE         PICTURE ZZZZ.ZZ.
13 0      02  FILLER                      PICTURE X(5)   VALUE IS SPACES.
14 0      02  ANNUAL-DETAIL-LINE          PICTURE Z(5).ZZ.
15 0  01  HEADING-RECORD.
16 0      02  FILLER                      PICTURE X(50)  VALUE IS SPACES.
17 0      02  WEEKLY-HEADING-LINE         PICTURE A(6)   VALUE 'WEEKLY'.
18 0      02  FILLER                      PICTURE X(5)   VALUE IS SPACES.
19 0      02  MONTHLY-HEADING-LINE        PICTURE A(7)   VALUE 'MONTHLY'.
20 0      02  FILLER                      PICTURE X(6)   VALUE IS SPACES.
21 0      02  ANNUAL-HEADING-LINE         PICTURE A(6)   VALUE 'ANNUAL'.
```

Figure 43. Sample Program #3 (part 2 of 5)



IBM

COBOL PROGRAM SHEET

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

PAGE 1 3 | PROGRAM: SAMPLE PROGRAM #3 | SYSTEM: 1460 | SHEET 3 OF 5

PROGRAMMER | DATE | IDENT. 73 SAMPLE-3 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

```
01 0  01  CORRECT-MESSAGE.
02 0      02  FILLER                      PICTURE X(5)   VALUE IS SPACES.
03 0      02  TABLE-IS-CORRECT            PICTURE A(32)  VALUE IS
04 0          'TABLE VALUES ARE CORRECT'.
05 0  01  INCORRECT-MESSAGE.
06 0      02  FILLER                      PICTURE X(512) VALUE IS SPACES.
07 0      02  TABLE-IS-NOT-CORRECT        PICTURE A(28)  VALUE IS
08 0          'TABLE VALUES ARE NOT CORRECT'.
09 0  77  HASH-TOTAL-COUNTER-WEEKLY        PICTURE 9(6)V999 VALUE IS ZERO.
10 0  77  HASH-TOTAL-COUNTER-MONTHLY       PICTURE 9(6)V999 VALUE IS ZERO.
11 0  77  HASH-TOTAL-COUNTER-ANNUAL        PICTURE 9(6)V999 VALUE IS ZERO.
12 0  77  WEEKLY-PAY                       PICTURE 999V999.
13 0  77  MONTHLY-PAY                      PICTURE 9999V999.
14 0  77  ANNUAL-PAY                       PICTURE 9(5)V999.
15 0  CONSTANT SECTION.
16 0  77  HASH-TOTAL-OF-WEEKLY-PAY         PICTURE 9(6)V999 VALUE 008826.69.
17 0  77  HASH-TOTAL-OF-MONTHLY-PAY        PICTURE 9(6)V999 VALUE 038250.00.
18 0  77  HASH-TOTAL-OF-ANNUAL-PAY         PICTURE 9(6)V999 VALUE 459000.00.
```

Figure 43. Sample Program #3 (part 3 of 5)

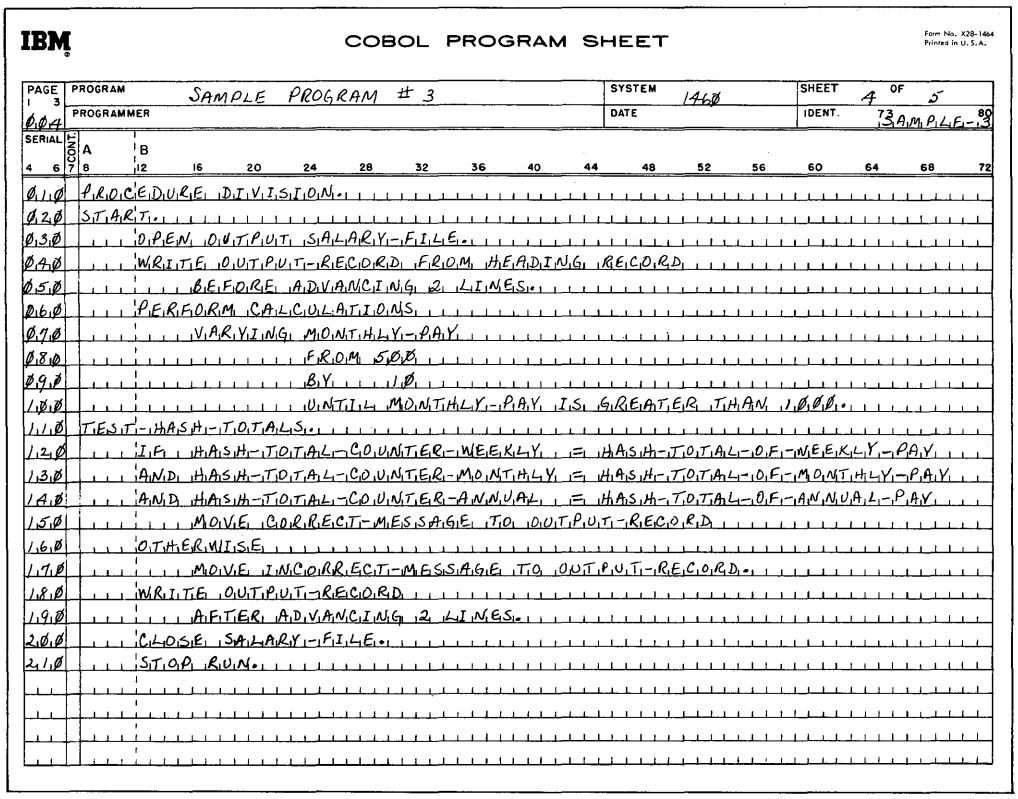


Figure 43. Sample Program #3 (part 4 of 5)

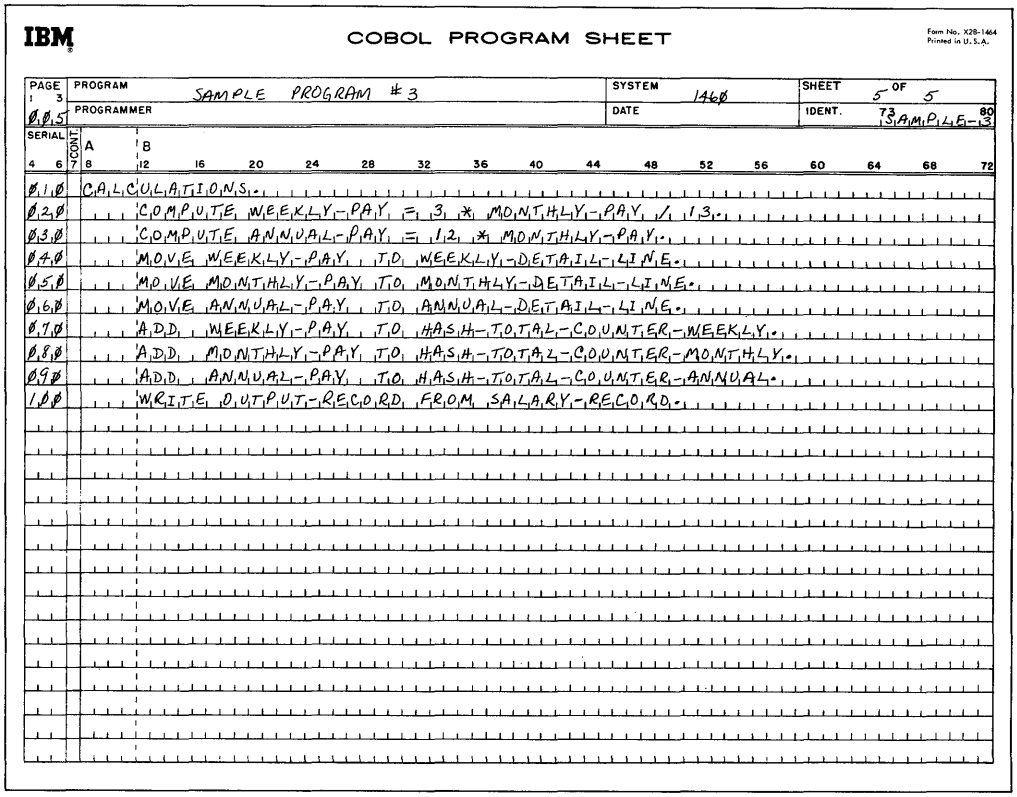


Figure 43. Sample Program #3 (part 5 of 5)

Programming Considerations

Aids

Two aids to generating more efficient machine language coding and decreasing compiling time are the optional WORK4 and WORK5 file assignments [COBOL *(on Disk) Program Specifications and Operating Procedures, IBM 1401, 1440, and 1460,* C24-3242].

The use of WORK4 intersperses COBOL source statements, by paragraph, with the Autocoder symbolic statements generated by the COBOL compiler. The programmer can then determine which autocoder statements were generated for the respective COBOL statements.

The use of WORK5 produces a listing of the Autocoder symbolic statements generated by the COBOL compiler. It is valuable when warning diagnostics are generated. Errors can be corrected before the generated autocoder program is assembled, thus saving the extra assembly time.

Techniques

COBOL provides a convenient method of writing business-oriented programs. However, certain techniques can be used to produce more efficient machine language coding and increased compiling speed.

The following considerations and suggestions are included to aid the programmer in obtaining a better COBOL-generated program. An original program (Figure 43) required approximately 3,100 positions of core storage. By applying a few of the suggestions to the second program (Figure 44) the core storage requirement is reduced to approximately 2,350 positions of core storage, representing a saving of 25 percent.

The changed statements utilize redefinition, equal decimal alignment, alphabetic compare, and the deletion of a subroutine caused by the statement WRITE OUTPUT-RECORD FROM SALARY-RECORD (Figure 43, part 5 of 5, line 100). It is recommended that the programmer become familiar with these suggestions and apply them in the writing of COBOL programs.

Area Allocation in the Data Division

The following rules govern when 1401 COBOL sets word marks with data areas:

1. Record areas (01 entries) always have a group mark with a word mark in the following position, and have a word mark in the high order position.
2. Word marks will be set in the high order positions at the next level from the 01 entry. This will be 02, or the next lower level if no 02 is present, unless occurs or redefinition is present.
3. Subfields have word marks set only when their high order positions coincide with word marks set as in preceding item 2.
4. A word mark is always set in the high order position at the 77 levels, but there is no group mark with a word mark set.
5. No word marks are set for data fields within a 01 entry which contains a redefines or an occurs, either at the 01 entry (implicit redefinition is allowable) or at any sublevel.

If word marks are required but not present, they will be set continually and cleared for access to the field; this requires time and core. If word marks are present, they will be regenerated if removed. For example, if editing into a 02 area, a word mark will be reset each time.

Tables

Many programs require tables. Following are several considerations about table building and searching with 1401 COBOL.

1. Unless it is certain that a table will never change, the initial values in the table should not be established with the VALUE clause. A better approach is to set up a card deck or tape file with one table entry and a sequence number on each record. Using the READ verb, build up the table data during program initialization. This approach eliminates the need for recompilation or object-program patching in the event that the table changes in value or size.
2. Before using the OCCURS clause and one or more levels of subscripting, weigh the alternate storage cost of naming each table entry and writing (for example):

   IF ARG = TAB-1 MOVE ENT-1 TO WORK GO TO FOUND.
   IF ARG = TAB-2 MOVE ENT-2 TO WORK GO TO FOUND.
   etc.
3. Define long tables as a set of shorter tables. A few IF statements are enough to isolate the relevant position, which can then be moved to a work area where the final pinpointing of the correct entry can be done.
4. If the work area mentioned in the preceding item 3 is n entries long where n is a power of 2 (such as 8 or 16), the IF statements which are used can be written in such a way as to effect a binary search. In the case of a 16-entry work area, this technique can yield an answer after only four IF statements.
5. Sequential table searches require little programming effort and are efficient if the table can be arranged so that the most active items are at the beginning of the table.

Move Verb

1. MOVE A TO B, where A and B are equal length alphanumeric elementary items defined at either the 01 or 02 levels, gives the best possible coding.

All items with subfields are treated as alphanumeric by COBOL, even if some or all subfields are defined as numeric. Only one 7 character instruction is generated as long as A and B are not redefined or subscripted.

2. If both A and B are redefined items or items defined at 03 levels and up, eight additional characters of instructions are generated (i.e. SET WORD MARK and CLEAR WORD MARK).
3. Elementary items are treated as above unless they have an unequal number of decimal places. In that case, a greater number of instructions is generated.
4. Unequal length elementary alphanumeric items are moved the same as equal length items when A is longer than B. However when B is longer, additional instruction characters are generated to blank the receiving field.
5. MOVE A TO B causes COBOL to include a special subroutine when A and B are of unequal length or one or both contain subfields. The special subroutine is used because the MLC and MRCM instructions cannot conveniently handle this complex situation. Even when A and B are the same length, the subroutine is still used if A is a 01 item and B is a 77 item or vice versa. The subroutine may be avoided by writing a set of individual MOVES, redefining both A and B, or by making them the same length.
6. MOVE SPACES TO A and MOVE ZEROS TO A each generate 11 characters of object code unless A is a 01 level item with subfields. In that case, A can be redefined at an additional cost of eight characters of object code.
7. When editing is involved in MOVE A TO B, the same rules about scaling, redefinition, and size apply. For example, when the A field has fewer decimal places than the editing PICTURE describing B, many characters of coding are generated. If the scaling is identical for A and B, approximately one-third as many instruction characters are generated, plus the edit word.
8. Avoid editing functions which cannot be handled by the edit instruction directly; COBOL zeros, floating plus or minus, DB, and single plus. A special subroutine is called to handle these cases.
9. MOVE ALL requires a special subroutine. Use a literal or constant of correct length to handle this case.

If Statement

1. When defining fields that are to be compared, consider the following:
   a. When at least one of the fields is a 01 item with subfields, a special subroutine is required. It is better to process such fields by comparing each lower-level item individually; or the group item can be moved to a hold area of equal size (not containing subfields), and then comparing.
   b. When numeric compares must be used because one or both of the fields are signed, attempt to arrange the record format so each item has the same number of decimal places. The fields do not have to be the same total length.
2. In the statement IF A = B, only one of the fields (A or B) need be defined as alphanumeric to get the more efficient alphanumeric compare instructions generated.
3. IF A NOT GREATER THAN B . . . has the same meaning as IF A LESS THAN B OR EQUAL TO B . . . and the generated instructions for the first statement require half the number of core positions.
4. The statement IF A IS ZERO . . . generates more efficient coding when A is defined as numeric rather than alphanumeric. However, an even greater improvement can be gained by declaring a constant of zeros (named C, for example), and writing IF A = C . . . which is twice as fast.
5. Avoid the statements IF A ALPHABETIC and IF A NUMERIC whenever possible because they require subroutines in the object program.
6. Avoid the use of ALL, HIGH-VALUES, LOW-VALUES, SPACES, and ZEROS in conditional expressions. They can easily be replaced by named constants.
7. Subscripted names in an IF statement will cause the compiler to include appropriate subroutines which often perform slowly at object time. Frequently it is better to use several IF statements to perform a table look-up on a short table rather than use subscripting and the PERFORM verb (or an equivalent loop).

Arithmetic Verbs

1. Avoid ON SIZE ERROR . . . whenever possible. The generated coding to perform this test consists of up to 40 characters.
2. ADD and SUBTRACT statements:
   a. The most efficient object coding is obtained for fields which have equal scaling. When two fields (A and B) have equal scaling, the statement ADD A to B generates 7 characters of object code.
   b. Redefining, or using 03 levels or greater, will require 8 additional characters for each field so defined.
   c. Multiple operands are as efficient as the equivalent set of single statements. ADD A, B TO C generates 14 characters (assuming the requirements of 3a are met).
   d. ADD A TO A is an economical way of multiplying A by two. Other sequences of ADD's and SUBTRACT's, sometimes with REDEFINE's to achieve a

shift, can be devised to simulate a more complex multiplication.

3. MULTIPLY and DIVIDE statements:
   a. MULTIPLY A BY B GIVING C generates 21 characters of instructions if A, B, and C have no decimal places. When A, B, and C have decimals, and the number of decimals in C is not the sum of those in A and B, 42 characters of instructions are generated.
   b. In the preceding example, ROUNDED generates an additional 7 characters.
   c. Less efficient coding is generated for a COMPUTE statement than for the equivalent set ADD, SUBTRACT, MULTIPLY, and DIVIDE statements. The reason for this is the need to retain up to 18-digit precision throughout the execution of a COMPUTE statement. Because the 18 digits can be on either side of the decimal point, and because one or two extra digits may be required for rounding, COBOL allocates 40 digit accumulators for the storage of temporary results.

      Work areas are assigned only once per program. Thus the most complex COMPUTE statement determines the number of 40 character areas that will be needed for *all* COMPUTE's.

Perform and Alter Statements

1. The statement ALTER LABEL TO PROCEED TO NEXT-LABEL generates 10 characters of coding.
2. The statement PERFORM CALCULATION generates 18 characters of coding at the point in the program where the PERFORM occurs. In addition, CALCULATION is augmented by 4 positions for each PERFORM which references it.
3. CALCULATION should be positioned in the source program at the point where it will be executed most frequently simply by falling through from the preceding paragraph.
4. The option 2 statement, PERFORM CALCULATION 5 TIMES is efficient. Core requirements are about 45 positions at the point in the program where the PERFORM occurs and 4 positions additional at the end of CALCULATION. No additional core or time is required when a data-name instead of a literal is used to indicate the number of TIMES.
5. Option 4 of the PERFORM verb is handled best if the VARYING field is defined as alphanumeric and each of the fields in the expression has the same length.

Input/Output Verbs

1. The statements READ INTO and WRITE FROM each cause a move of the entire logical record. In many cases the use of these options is unnecessary because processing can be done either in an input or an output record area as defined by the DATA RECORDS ARE clause in the FD's. When READ INTO or WRITE FROM must be used, ensure that the implied data move involves equal length areas.
2. When using a card reader, READ is faster and generally smaller than ACCEPT. Similarly, WRITE is better than DISPLAY for printing and punching.
3. It is not possible within COBOL to assign the same input/output area to two files. Areas in the WORKING-STORAGE SECTION can be (and should be) shared, however.
4. For card and printer files, input/output areas in addition to 001-080, 101-180, and 201-332 are assigned. This is in anticipation of a possible conflict with the ACCEPT and DISPLAY verbs, which use those areas also.
5. The WRITE verb for a printer FD does not clear the print area. Use MOVE SPACES to clear this area.
6. Form 3 (unblocked, variable length) tape records are not permitted within COBOL. If necessary the file can be defined as Form 1, and a simple Autocoder sequence can be used to set and clear the GMWM at the end of the portion of data to be written. Form 4 usually offers better tape utilization.

   A common error in COBOL programming is the assumption that a different area in WORKING-STORAGE must be defined for each record type in a given file. This may be avoided by (1) defining all possible data records directly under the FD with one 01 entry group per record type, or (2) defining the most common record type under the FD and all the others in a *single area* in WORKING-STORAGE which is redefined once for each record type.

Optional COBOL Words

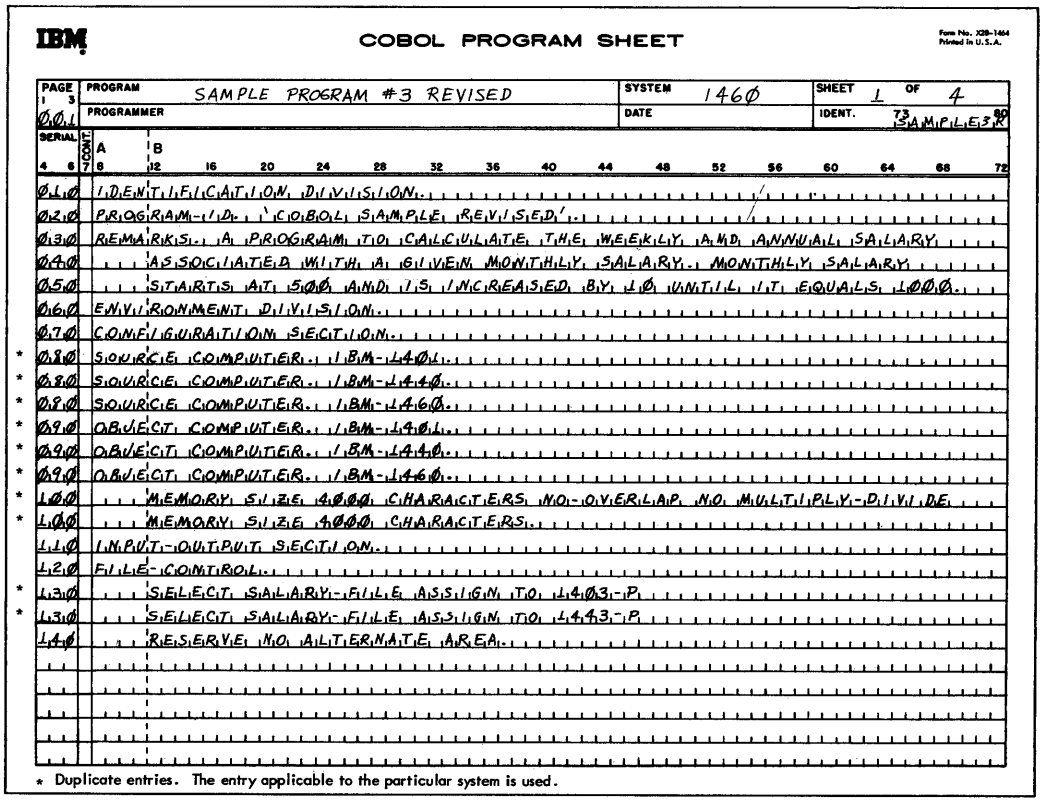
COBOL words, defined as being optional words in this manual, add nothing to the object program but do require time for the compiler to evaluate. Compiling time can be decreased by avoiding these optional COBOL words.

Object Time Subroutines

There are several COBOL object time subroutines that may be generated. These routines are described in a separate bulletin which may be obtained with the program. Normally, the programmer should avoid COBOL statements which cause these subroutines to be used. For the most part their inclusion is caused by either unusual language features or by complex data formats. Following is a list of these subroutines and the reason

why they are called and/or how they may be avoided.

1. The Examine subroutine is included whenever the EXAMINE verb is used. It may be avoided as follows:
   a. For short fields, give each position a name by defining an appropriate number of subfields and using a set of IF statements.
   b. For long fields, define a work area with one-character subfields and process portions of the long field there.
2. Single, double, and triple subscript subroutines are included whenever a field is singly, doubly, or three-level subscripted.
3. The Alpha Compare subroutine is included when a group item with subfields is compared to any data item. The subroutine may be avoided by redefining the field which contains subfields.
4. The Figcon Compare subroutine is included whenever a record with subfields is compared to a figurative constant (HIGH-VALUE, LOW-VALUE, QUOTE, and ALL alpha-literal). This subroutine may be avoided by redefining the field with subfields and using a literal or constant.
5. The If Numeric subroutine is included whenever an alphanumeric field whose size is greater than 1 is tested for a numeric value.
6. The If Alphabetic subroutine is included whenever an alphanumeric field whose size is greater than 1 is tested for an alphabetic value.
7. The Accept subroutine is included whenever the ACCEPT verb is used. To avoid this subroutine, define a file and use the READ verb.
8. The Display subroutine is included whenever the DISPLAY verb is used. To avoid this subroutine, define a file and use the WRITE verb.
9. The Editing subroutine is included when editing requirements include COBOL zero, floating + and − sign, single plus, and DB. It produces highly specialized editing features. If possible, use only the standard editing features of the 1401, 1440, or 1460.
10. The Expin subroutine is included whenever an integer exponent is used (COMPUTE A = B**5). It may be avoided by writing successive MULTIPLY's.
11. The Go To Depending subroutine is included whenever GO TO DEPENDING is used. This subroutine may be avoided by a set of IF statements.
12. The Move All subroutine is included when the ALL option of the MOVE verb is used and a record with subfields is to be filled. A MOVE statement or a set of MOVE statements is preferable.
13. The Move Record subroutine is included whenever a record with subfields is used in a MOVE statement, except when the other field is a record (01 level) of equal length. This subroutine may be avoided by:
    a. Using a set of elementary MOVE's.
    b. Redefining both fields to eliminate word marks.
14. The Expni subroutine is included when raising an expression by a non-integral exponent (COMPUTE A = B**2.5). It is impossible to perform all the functions of this subroutine with other COBOL statements unless the exponent is defined as an integer. For special purposes an Autocoder subroutine may be a more practical solution.
15. The Multiply subroutine (MULTY) is included whenever the object computer does not have the Multiply/Divide feature. The subroutine may be avoided by substituting a comparable set of ADD instructions.
16. The Stop-literal subroutine (SPLIT) is included whenever a STOP literal statement is used. The size of the subroutine may be reduced by declaring NO-CONSOLE-PRINTER.



IBM

COBOL PROGRAM SHEET

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

PAGE 1 3 | PROGRAM SAMPLE PROGRAM #3 REVISED | SYSTEM 1460 | SHEET 1 OF 4

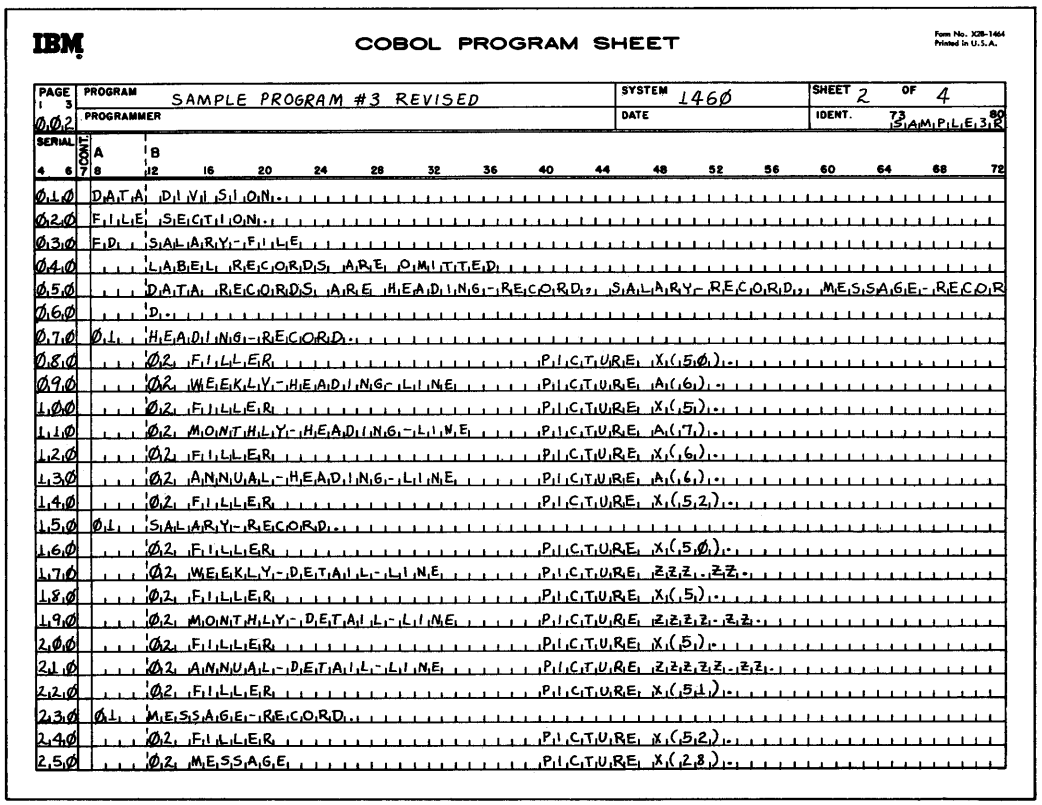
00.1 | PROGRAMMER | DATE | IDENT. 73 SAMPLE3R 80

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

```
01.0  IDENTIFICATION DIVISION.
02.0  PROGRAM-ID. 'COBOL SAMPLE REVISED'.
03.0  REMARKS. A PROGRAM TO CALCULATE THE WEEKLY AND ANNUAL SALARY
04.0      ASSOCIATED WITH A GIVEN MONTHLY SALARY. MONTHLY SALARY
05.0      STARTS AT 500 AND IS INCREASED BY 10 UNTIL IT EQUALS 1000.
06.0  ENVIRONMENT DIVISION.
07.0  CONFIGURATION SECTION.
*  08.0  SOURCE COMPUTER. IBM-1401.
*  08.0  SOURCE COMPUTER. IBM-1440.
*  08.0  SOURCE COMPUTER. IBM-1460.
*  09.0  OBJECT COMPUTER. IBM-1401.
*  09.0  OBJECT COMPUTER. IBM-1440.
*  09.0  OBJECT COMPUTER. IBM-1460.
*  10.0      MEMORY SIZE 4000 CHARACTERS NO-OVERLAP NO MULTIPLY-DIVIDE
*  10.0      MEMORY SIZE 4000 CHARACTERS.
   11.0  INPUT-OUTPUT SECTION.
   12.0  FILE-CONTROL.
*  13.0      SELECT SALARY-FILE ASSIGN TO 1403-P
*  13.0      SELECT SALARY-FILE ASSIGN TO 1443-P
   14.0      RESERVE NO ALTERNATE AREA.
```

* Duplicate entries. The entry applicable to the particular system is used.

Figure 44. Sample Program #3 Revised (Part 1 of 4)



IBM

COBOL PROGRAM SHEET

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

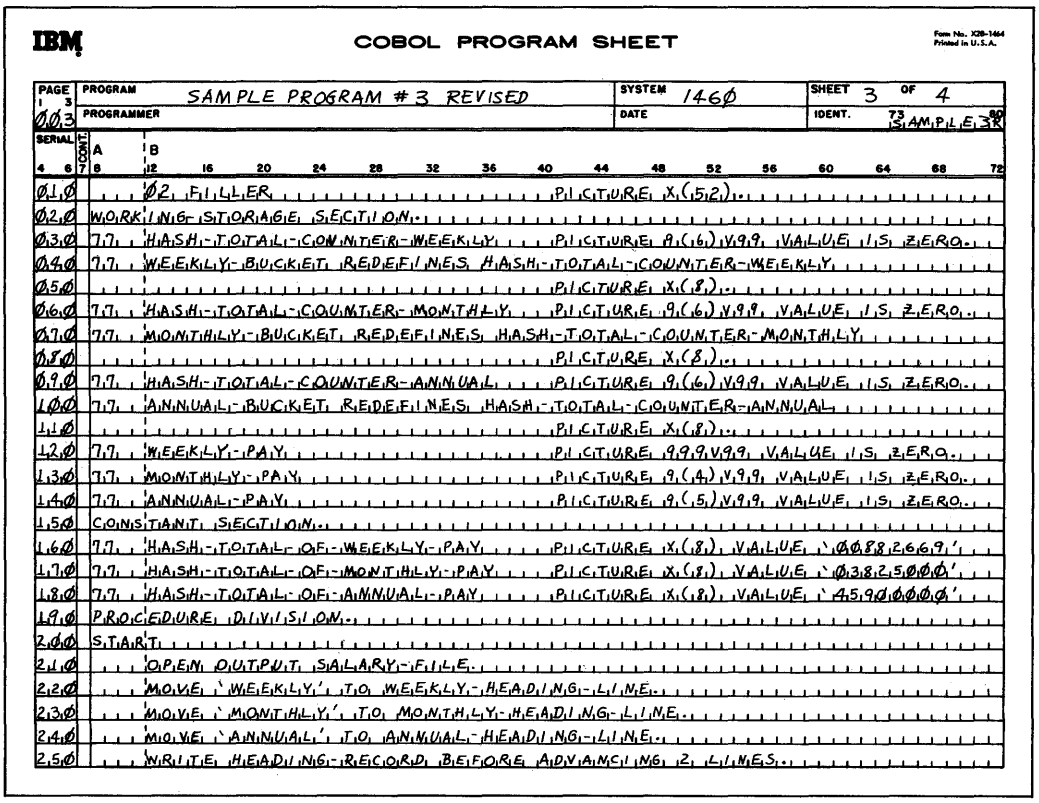
PAGE 2 3 | PROGRAM SAMPLE PROGRAM #3 REVISED | SYSTEM 1460 | SHEET 2 OF 4

00.2 | PROGRAMMER | DATE | IDENT. 73 SAMPLE3R 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

```
01.0  DATA DIVISION.
02.0  FILE SECTION.
03.0  FD  SALARY-FILE
04.0      LABEL RECORDS ARE OMITTED
05.0      DATA RECORDS ARE HEADING-RECORD, SALARY-RECORD, MESSAGE-RECOR
06.0      D.
07.0  01  HEADING-RECORD.
08.0      02  FILLER                  PICTURE X(50).
09.0      02  WEEKLY-HEADING-LINE     PICTURE A(61).
10.0      02  FILLER                  PICTURE X(51).
11.0      02  MONTHLY-HEADING-LINE    PICTURE A(71).
12.0      02  FILLER                  PICTURE X(61).
13.0      02  ANNUAL-HEADING-LINE     PICTURE A(61).
14.0      02  FILLER                  PICTURE X(52).
15.0  01  SALARY-RECORD.
16.0      02  FILLER                  PICTURE X(50).
17.0      02  WEEKLY-DETAIL-LINE      PICTURE ZZZ.ZZ.
18.0      02  FILLER                  PICTURE X(5).
19.0      02  MONTHLY-DETAIL-LINE     PICTURE ZZZZ.ZZ.
20.0      02  FILLER                  PICTURE X(5).
21.0      02  ANNUAL-DETAIL-LINE      PICTURE ZZZZZ.ZZ.
22.0      02  FILLER                  PICTURE X(51).
23.0  01  MESSAGE-RECORD.
24.0      02  FILLER                  PICTURE X(52).
25.0      02  MESSAGE                 PICTURE X(28).
```

Figure 44. Sample Program #3 Revised (Part 2 of 4)



IBM

COBOL PROGRAM SHEET

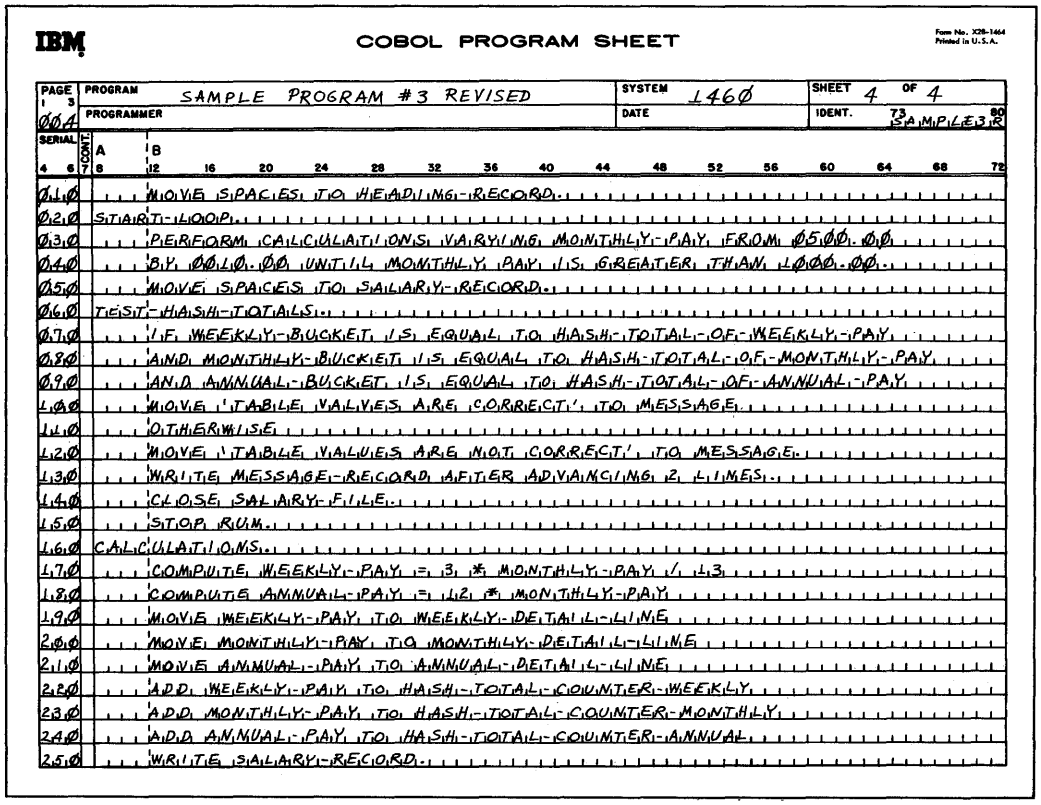
Form No. X28-1464 Printed in U.S.A.

PAGE 3 | PROGRAM: SAMPLE PROGRAM #3 REVISED | SYSTEM: 1460 | SHEET 3 OF 4
ØØ3 | PROGRAMMER: | DATE: | IDENT. 73 SAMPLE3R 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

```
Ø1Ø    Ø2 FILLER                    PICTURE X(1512).
Ø2Ø  WORKING-STORAGE SECTION.
Ø3Ø   77  HASH-TOTAL-COUNTER-WEEKLY    PICTURE 9(16) V999 VALUE IS ZERO.
Ø4Ø   77  WEEKLY-BUCKET REDEFINES HASH-TOTAL-COUNTER-WEEKLY
Ø5Ø                                   PICTURE X(8).
Ø6Ø   77  HASH-TOTAL-COUNTER-MONTHLY   PICTURE 9(6) V999 VALUE IS ZERO.
Ø7Ø   77  MONTHLY-BUCKET REDEFINES HASH-TOTAL-COUNTER-MONTHLY
Ø8Ø                                   PICTURE X(8).
Ø9Ø   77  HASH-TOTAL-COUNTER-ANNUAL    PICTURE 9(6) V999 VALUE IS ZERO.
1ØØ   77  ANNUAL-BUCKET REDEFINES HASH-TOTAL-COUNTER-ANNUAL
11Ø                                   PICTURE X(8).
12Ø   77  WEEKLY-PAY                  PICTURE 999V99 VALUE IS ZERO.
13Ø   77  MONTHLY-PAY                 PICTURE 9(4) V99 VALUE IS ZERO.
14Ø   77  ANNUAL-PAY                  PICTURE 9(5) V99 VALUE IS ZERO.
15Ø  CONSTANT SECTION.
16Ø   77  HASH-TOTAL-OF-WEEKLY-PAY    PICTURE X(8) VALUE 'ØØ88266917'.
17Ø   77  HASH-TOTAL-OF-MONTHLY-PAY   PICTURE X(8) VALUE 'Ø38255ØØØØ'.
18Ø   77  HASH-TOTAL-OF-ANNUAL-PAY    PICTURE X(8) VALUE '45ØØØØØØØ'.
19Ø  PROCEDURE DIVISION.
2ØØ  START.
21Ø      OPEN OUTPUT SALARY-FILE.
22Ø      MOVE 'WEEKLY' TO WEEKLY-HEADING-LINE.
23Ø      MOVE 'MONTHLY' TO MONTHLY-HEADING-LINE.
24Ø      MOVE 'ANNUAL' TO ANNUAL-HEADING-LINE.
25Ø      WRITE HEADING-RECORD BEFORE ADVANCING 2 LINES.
```

Figure 44. Sample Program #3 Revised (Part 3 of 4)



IBM

COBOL PROGRAM SHEET

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

PAGE 3 | PROGRAM: SAMPLE PROGRAM #3 REVISED | SYSTEM: 1460 | SHEET 4 OF 4
ØØ4 | PROGRAMMER: | DATE: | IDENT. 73 SAMPLE3R 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

```
Ø1Ø      MOVE SPACES TO HEADING-RECORD.
Ø2Ø  START-LOOP.
Ø3Ø      PERFORM CALCULATIONS VARYING MONTHLY-PAY FROM Ø5ØØ.ØØ
Ø4Ø      BY ØØ1Ø.ØØ UNTIL MONTHLY-PAY IS GREATER THAN 1ØØØ.ØØ.
Ø5Ø      MOVE SPACES TO SALARY-RECORD.
Ø6Ø      TEST-HASH-TOTALS.
Ø7Ø          IF WEEKLY-BUCKET IS EQUAL TO HASH-TOTAL-OF-WEEKLY-PAY
Ø8Ø          AND MONTHLY-BUCKET IS EQUAL TO HASH-TOTAL-OF-MONTHLY-PAY
Ø9Ø          AND ANNUAL-BUCKET IS EQUAL TO HASH-TOTAL-OF-ANNUAL-PAY
1ØØ              MOVE 'TABLE VALUES ARE CORRECT' TO MESSAGE
11Ø          OTHERWISE
12Ø              MOVE 'TABLE VALUES ARE NOT CORRECT' TO MESSAGE.
13Ø      WRITE MESSAGE-RECORD AFTER ADVANCING 2 LINES.
14Ø      CLOSE SALARY-FILE.
15Ø      STOP RUN.
16Ø  CALCULATIONS.
17Ø      COMPUTE WEEKLY-PAY = 3 * MONTHLY-PAY / 13.
18Ø      COMPUTE ANNUAL-PAY = 12 * MONTHLY-PAY.
19Ø      MOVE WEEKLY-PAY TO WEEKLY-DETAIL-LINE
2ØØ      MOVE MONTHLY-PAY TO MONTHLY-DETAIL-LINE
21Ø      MOVE ANNUAL-PAY TO ANNUAL-DETAIL-LINE
22Ø      ADD WEEKLY-PAY TO HASH-TOTAL-COUNTER-WEEKLY
23Ø      ADD MONTHLY-PAY TO HASH-TOTAL-COUNTER-MONTHLY
24Ø      ADD ANNUAL-PAY TO HASH-TOTAL-COUNTER-ANNUAL
25Ø      WRITE SALARY-RECORD.
```

Figure 44. Sample Program #3 Revised (Part 4 of 4)

Index

IBM 1401, 1440, 1460 Printed in U.S.A. C24-3235-2

IBM

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