File Number 1401-24
Form C24-1492-2

IBM Systems Reference Library

COBOL (on Tape) Specifications IBM 1401

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 Data Processing System.

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

A sample problem shows complete entries for the IDENTIFICATION, ENVIRONMENT and DATA DIVISIONS.

Preface

This publication partially describes the specifications for writing a 1401 COBOL program to be processed by an IBM 1401 with at least 4,000 positions of core storage. It supplements the IBM COBOL General Information Manual, Form F28-8053, which describes the COBOL language as it applies to IBM computers. The manual includes many details about the COBOL language that are not included in this publication.

The 1401 COBOL programmer should also be familiar with the material contained in the following Systems Reference Library publications:

IBM 1401 System Operation Reference Manual, Form A24-3067

Autocoder (on Tape) Language Specifications and Operating Procedures IBM 1401 and 1460, Form C24-3319

Input/Output Control System Specifications and Operating Procedures for IBM 1401 and 1460, Form C24-1462

Operating information for this version of 1401 COBOL is in *COBOL (on Tape) Operating Procedures IBM 1401,* Form C24-3146.

Since the IBM 1401 Data Processing System was announced, the IBM 1401 Symbolic Programming System has been widely used. This system is essentially a one-for-one coding system in which the programmer writes one symbolic instruction for each actual machine-language instruction needed to solve a given problem. This system relieves the programmer from the burden of remembering machine-language operation codes and machine addresses.

The 1401 Autocoder was subsequently developed to permit the programmer to use macro instructions that incorporate common routines automatically. The 1401 Input/Output Control System (IOCS) has been included in Autocoder to provide standard routines for getting data into and out of the machine.

The COBOL language permits programming in terms that describe the problem, rather than in terms that describe the machine used to solve it. The general language specifications (with a few minor exceptions) remain the same for all computers.

To write a complete COBOL program for the 1401, the reader must be familiar with the information presented in the COBOL General Information Manual as well as the material in this publication.

This publication, C24-1492-2, is a major revision of C24-1492-1 and obsoletes it and prior editions. In addition to incorporating information released in Technical Newsletters N24-0259 and N21-0033, significant changes have been made. A section on Programming Considerations has been added.

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 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 1401 COBOL processor.

The Cobol Processor

The COBOL processor is itself a program. It compiles an object program in 1401 Autocoder language from the COBOL source statements. The Autocoder processor assembles the machine-language object program from the object program in Autocoder as shown in Figure 1.

Machine Requirements

The 1401 COBOL processor can compile an object program for any IBM 1401 system that has at least 4,000 positions of core storage. However, to process the COBOL source program, the 1401 must have at least:

- 4,000 positions of core storage
- Four IBM magnetic-tape units
- IBM 1403 Printer, Model 2
- IBM 1402 Card Read-Punch
- Advanced Programming Feature
- High-Low-Equal Compare Feature
- Sense Switches

The 1401 on which the object program is to be executed must have:

1. Sufficient core storage to contain the program produced 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.

   NOTE: This requirement is a significant consideration when planning to implement COBOL on a system with 4000 positions of core storage.

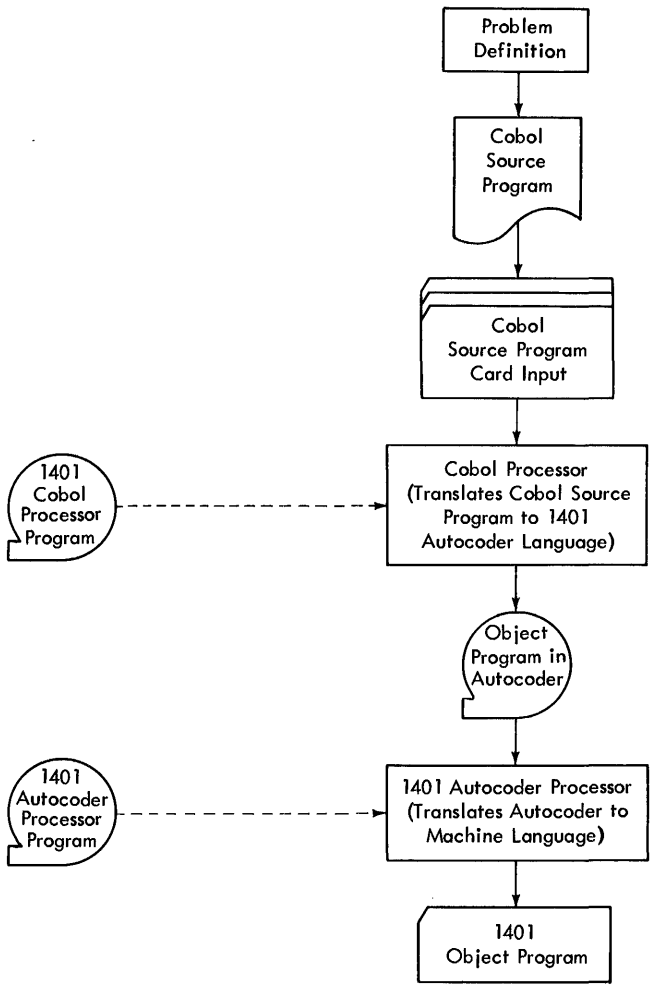


Figure 1. COBOL Compiling and Assembly Process

2. The object machine must have the input and output units defined in the FILE-CONTROL paragraph.
3. Advanced Programming Feature.
4. High-Low-Equal Compare Feature.
5. Sense Switches when they are referred to in the SPECIAL-NAMES section.
6. Multiply-Divide Feature if any of these entries appears in the PROCEDURE DIVISION of the COBOL source program:
   a. MULTIPLY verb.
   b. DIVIDE verb.
   c. COMPUTE verb when either /,*, or ** is used as the operator.

The 1401 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,* Form F28-8053. These division statement sets must be arranged for presentation to the 1401 COBOL processor in this order:

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

Write the IDENTIFICATION DIVISION entries 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 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 COBOL presentation format for this is:

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

Configuration Section

The CONFIGURATION SECTION has three paragraphs: The SOURCE-COMPUTER paragraph names and describes the 1401 that will compile the object program from the COBOL source statements.

The OBJECT-COMPUTER paragraph names and describes the 1401 that will execute the object program.

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

Source-Computer Paragraph

Reference Format

SOURCE-COMPUTER. IBM-1401

[MEMORY SIZE { 4000 / 8000 / 12000 / 16000 } CHARACTERS]

[NO-RELEASE] [NO-PRINT-STORAGE].

General Description: This paragraph names the computer that will compile and assemble the object program. It is the computer in which the IBM 1401 COBOL processor program compiles a machine-oriented symbolic program (1401 Autocoder) from the problem-oriented COBOL source program and assembles the actual machine-language program.

[MEMORY SIZE { 4000 / 8000 / 12000 / 16000 } CHARACTERS]

General Description: This statement tells the COBOL processor how much core storage (memory) is available for use during the compiling and assembling operation. If this statement is omitted, the actual machine core-storage size will be used. If the clause is included, the specified machine size will be used unless it is greater than the actual machine core-storage size.

[NO-RELEASE] [NO-PRINT-STORAGE].

The appropriate clause(s) must be included if the source computer does not have the read-punch release or print-storage features.

Object-Computer Paragraph

Reference Format

```
OBJECT-COMPUTER.    IBM-1401
    [ ASSIGN OBJECT-PROGRAM TO TAPE ]
[ MEMORY SIZE { { 4000 } CHARACTERS                    } ]
              { { 8000 }                               }
              { {12000 }                               }
              { {16000 }                               }
              {   ADDRESS integer-1 THRU               }
              {           { 4000 }                     }
              {           { 8000 }                     }
              {           {12000 }                     }
              {           {16000 }                     }
    [ NO-OVERLAP ]
    [ NO-PRINT-STORAGE ] .
```

General Description: The OBJECT-COMPUTER paragraph describes the computer that will execute the object program. The "OBJECT-COMPUTER. IBM-1401" statement defines an IBM 1401 with 16,000 characters, the processing-overlap feature, and input/output units required for the files defined in the FILE-CONTROL paragraph. If the object machine has fewer than 16,000 positions of storage and/or no overlap, and/or no print storage, the appropriate clauses must be included in the source program. It also permits the programmer to specify which input device will read the object program into storage before it is executed.

```
[ ASSIGN OBJECT-PROGRAM TO TAPE ]
```

General Description: This statement directs the processor to put the object program on magnetic tape. If this statement is included in the COBOL source program, a tape unit is designated as the input device that will read the object program into storage. If the statement is omitted, the object program will be punched in cards and the IBM 1402 Card Read-Punch will be the input device.

```
[ MEMORY SIZE { { 4000 }      CHARACTERS       } ]
              { { 8000 }                       }
              { {12000 }                       }
              { {16000 }                       }
              {   ADDRESS integer-1 THRU       }
              {           { 4000 }             }
              {           { 8000 }             }
              {           {12000 }             }
              {           {16000 }             }
```

General Description: 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 333, he can use the ADDRESS *integer-1* THRU option and write the numerical address of this location in the *integer-1* portion. This number should not be less than 333. The number following THRU specifies the last core-storage position that can be used for the object program.

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 333.

```
[ NO-OVERLAP ]
```

General Description: The NO-OVERLAP option must be included only if the object computer does not have the processing overlap feature.

When the overlap feature is used, an alternate area must be specified for tape input files with form-1 records.

```
[ NO-PRINT-STORAGE ] .
```

General Description: This clause must be included if the object computer is not equipped with the print-storage feature.

Example: Figure 2 shows a sample OBJECT-COMPUTER paragraph.

```
CONT  A   B
7  8  12  16  20  24  28  32  36  40  44  48  52
OBJECT-COMPUTER.  IBM-1401
    ASSIGN OBJECT-PROGRAM TO TAPE
    MEMORY SIZE ADDRESS 438 THRU 12000
    NO-OVERLAP.
```

Figure 2. OBJECT-COMPUTER Paragraph

Special-Names Paragraph

Reference Format

```
SPECIAL-NAMES.

[ device-name-1 IS mnemonic-name-1 [ device-name-2 IS
  mnemonic-name-2 . . . ] ].

[ switch-name-1 [ ON STATUS IS condition-name-1 ]
[ OFF STATUS IS condition-name-2 ] ]
                [ switch-name-2 . . . ] ].

[ AUTOCODER-name IS COBOL-name [ AUTOCODER . . . ] ].
```

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

Device-Names

General Description: The standard device-names for the 1401 signal the COBOL processor which devices are available in the object computer. They are written with mnemonic-names the programmer has used to refer to them in the PROCEDURE DIVISION. The 1401 device-names are:

Device-Name	*Actual IBM 1401 Device*
1402-R, n	1402 Card Reader
1402-P, n	1402 Card Punch
1403-P	1403 Printer
1403-CT, n	1403 Printer Carriage

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 invalid or 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.

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.

For the 1403-CT 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 1403-CT, 3.

```
[ AUTOCODER-name IS COBOL-name ]
```

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 3 must appear in the SPECIAL-NAMES section of the COBOL program.

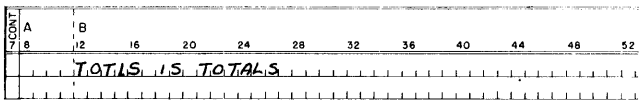


Figure 3. Equating and Autocoder-Name to a COBOL-Name

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 1401 switch-names are:

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

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

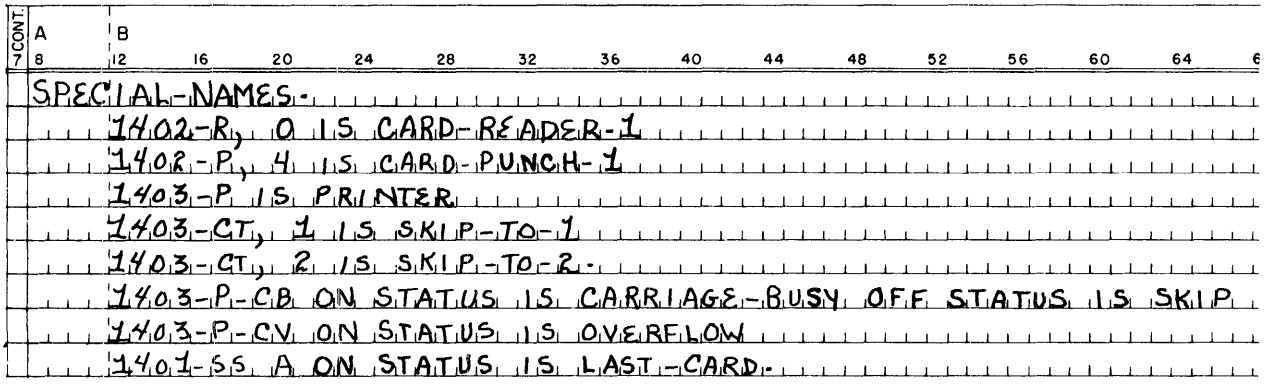


Figure 4. SPECIAL-NAMES Paragraph

The status of these switches may be interrogated by expressions in the PROCEDURE DIVISION that use condition-names.

Example: Figure 4 shows a sample SPECIAL-NAMES paragraph written for a 1401.

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.

The I-O-CONTROL paragraph is a deferred feature for 1401 COBOL and thus can not be included in a 1401 COBOL source program. See *Deferred Features.*

File-Control Paragraph

Reference Format

FILE-CONTROL. SELECT *file-name-1*
ASSIGN TO *device-name-1*
[RESERVE { 1 / NO } ALTERNATE AREA[S]] .

[SELECT . . . . .] .

General Description: 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 to specify an alternate input-output area for magnetic tape files if the 1401 has overlap processing.

SELECT *file-name-1*

General Description: 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 5):

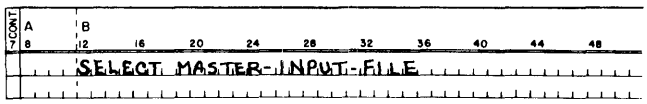


Figure 5. SELECT

ASSIGN TO *device-name-1*

General Description: Each file must be assigned to an input or output device-name. The 1401 device-names that are valid in the FILE-CONTROL paragraph are:

Device-Name	*Description*
TAPE(S) u a	Magnetic Tape Unit
1402-R, u	Card-Reader
1402-P, u	Card-Punch
1403-P	Printer

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. The *a* specifies that an alternate unit is to be assigned. It can be any digit from 1 to 6, 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 TAPE(s) and *u* and between *u* and *a*. (If the user specifies an invalid *u*, the processor substitutes the value 6.)

Example (Figure 6):

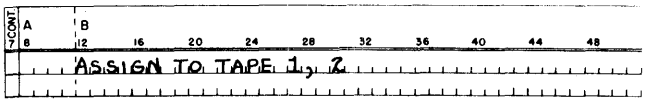


Figure 6. ASSIGN

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. Their device-names are written as described in *Special-Names Paragraph.*

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

General Description: 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 overlap feature.

Example (Figure 7):

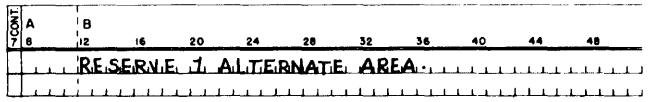


Figure 7. RESERVE

Figure 8 shows a sample FILE-CONTROL paragraph.

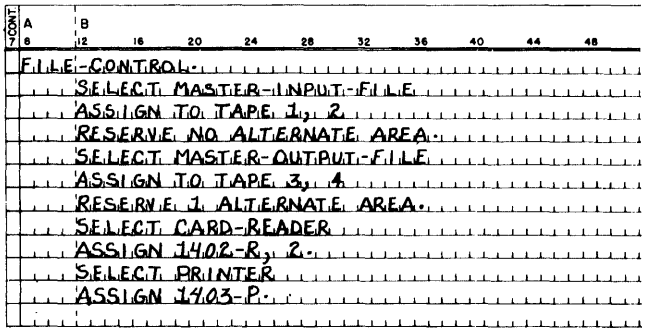


Figure 8. FILE-CONTROL Paragraph

Added Elective Elements—Environment Division

The MEMORY SIZE option of the SOURCE-COMPUTER paragraph is not contained in the COBOL General Information Manual, but is provided in the 1401 COBOL processor as it has been described in this publication.

Deferred Elements—Environment Division

Several elements are described in the COBOL General Information Manual that are not contained in this version of 1401 COBOL processor. These should not be coded in the ENVIRONMENT DIVISION entries for a 1401 COBOL program. They are stated here for reference:

1. The entire I-O-CONTROL paragraph (elective COBOL element).
2. The OPTIONAL option of the FILE-CONTROL paragraph.
3. The MULTIPLE REEL option in the FILE-CONTROL paragraph and all other features that provide for automatic assignments of tape units for a file.
4. The entire COPY option. (The library tape for the 1401 COBOL processor does not presently support the COPY feature.)
5. The RENAMING clause in the FILE-CONTROL paragraph.

IBM 1401 COBOL Tape Labels

The 1401 COBOL processor provides for IBM 1401 COBOL tape labels. These labels identify the file and specify the number of records in the file, the date it was created, and the length of time it must be kept. Two labels, a header and a trailer, are required for each labeled file.

IBM Header Labels

A header label is the first record of each reel of a file. It identifies the tape. The header label format is shown in Figure 9.

FIELD NO.	POSITION	CONTENTS	FIELD NAME	EXAMPLE
1	1-4 5	1HDR Blank	Header Label Identifier	1HDRb
2	6-10	5 Digits	Tape Serial Number	12345
3	11-15	5 Digits	File Serial Number	54321
4	16 17-19 20	- 3 Digits Blank	Reel Sequence Number	-002b
5	21-30	10 Characters	File Identification Name	PAYRLMASTR
6	31-35	5 Digits	Creation Date (00-99)(001-366) Year Day	63203
7	36 37-39 40	- 3 Digits Blank	Retention Cycle (000-365)	-007b
8	41-80	Not used		

Figure 9. COBOL Header Label Format

Header Label Identifier

These four characters indicate that the information contained in the record is the header label of a tape reel.

Tape Serial Number

These five digits identify the reel of tape within an installation. Each reel of tape should be given a tape serial number as soon as it is received at the installation. IOCS routines do not affect the tape serial number in a tape label.

File Serial Number

These five digits indicate a particular application or job number within an installation.

Reel Sequence Number

These three digits identify the reels in cases where multiple reels are needed for a specific job or application. The first reel is numbered 001 unless the user specifies another number.

File Identification Name

These ten characters identify the file. For example, PAYRLMASTR identifies the tape as the payroll master file.

Creation Date

These five digits contain the date on which the file was written originally. The two high-order digits indicate the year (00-99), and the remaining three digits indicate the nth day of that year (001-366).

Retention Cycle

These three digits indicate the number of days the file is to be kept after the date the file was originated. Files should be preserved until all output data produced from them has been used successfully as new input. This ensures that any file that requires this file as input can be reconstructed if necessary.

Header labels provide for a 365-day maximum retention cycle. If the file must be kept indefinitely, the programmer can specify this by putting the digits 99 in the two high-order positions of the creation-date field.

IBM 1401 COBOL Trailer Labels

The last information record in a tape reel is a trailer label. It indicates that the reel currently being processed is the last reel of a file or that more reels must be processed. Trailer labels are written after the last record in the reel has been processed.

The IBM COBOL trailer label format is shown in Figure 10.

Field No.	Positions	Contents	Field Name	Example
1	1-4	"IEOF or IEOR"	Trailer Label Identifier	IEOF
2	5 6-10 11-80	Blank 5 Digits Not used	Block Count	13430

Figure 10. IBM 1401 COBOL Trailer Label Format

Trailer Label Identifier

These four characters indicate that the information contained in the record is the trailer label of a tape reel.

Block Count

This field contains the number of blocks contained in the reel. A count is developed during processing and is entered in the trailer label record.

Record Formats for Tape Files

Detailed information about record formats is presented in the publication *Input/Output Control System: Specifications and Operating Procedures for IBM 1401 and 1460,* Form C24-1462. General information is presented in the following sections. Records for tape files may be as large as 999 characters.

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 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 Inter-Record Gap (IRG). Figure 11 shows an example of form-1 records with record marks.

Figure 12 shows a form-1 record 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 spaces (blanks) 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 blank record will be substituted if there are not enough data records to fill the last block.

Figure 13 shows a fixed-length, block tape record with record marks and padding. Each block contains four records.

Form-3 Records

Form-3 records (variable unblocked) are not permitted with 1401 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 characters 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 14 is the same). Figure 14 shows the record format for a form-4 record.

NOTE: When programming for form-4 or form-2 tape records, the record entry must allow a position for the record mark. For output records, the record mark must be moved into the record area before the record is written.

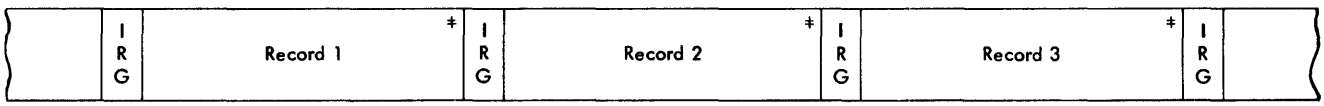


Figure 11. Form-1 Record with Record Marks

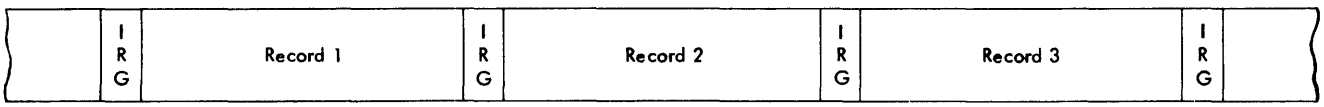


Figure 12. Form-1 Record without Record Marks

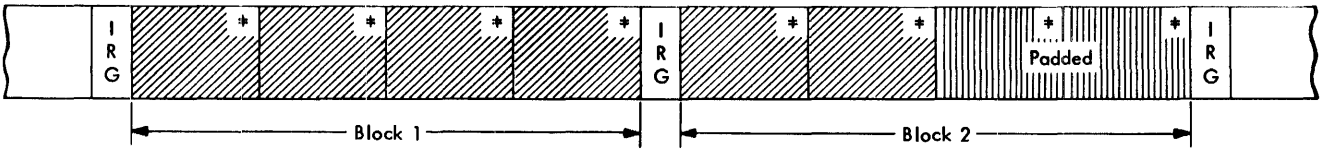


Figure 13. Form-2 Record with Record Marks

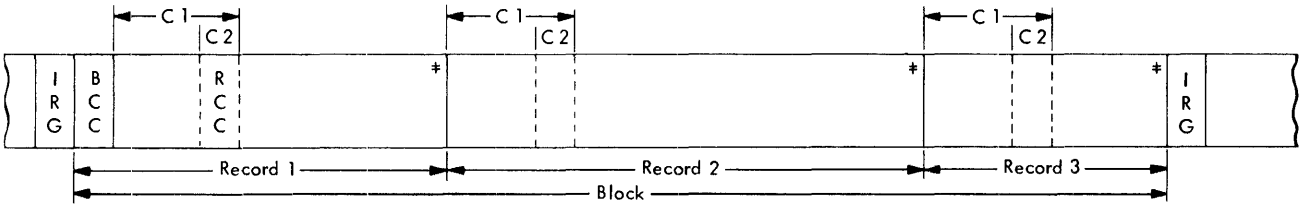


Figure 14. Form-4 Record

Record Formats for Punched-Card Files

Card Read-Punch Records

Records of files assigned to the 1402-R and the 1402-P 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 for tape files.

Printer Records

Records of files assigned to the 1403-P must also have form-1 record format. For the 1403 printer the fixed record size must be equal to the number of print positions on the printer (100 or 132).

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 1401 core storage where intermediate results and other items are stored temporarily at object-program execution time.

The CONSTANT SECTION describes fixed items of data that remain unchanged during the running of the object program. A date, for example, might be a fixed item, or constant.

The 1401 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-Description Entry

General Description: 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. A VALUE clause is required when label records are standard.

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 / IS } { STANDARD / OMITTED }

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

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

General Description: The level indicator identifies the beginning of the file-description entry and precedes the file name assigned by the programmer.

Example (Figure 15):

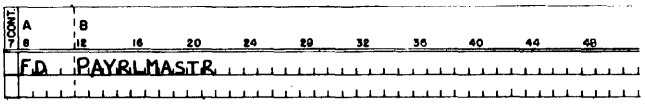


Figure 15. FD File-Name

```
[ RECORDING MODE IS 1 ]
```

General Description: 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 1401 COBOL.

If the RECORDING MODE clause is not included in the source program, the processor assumes recording mode 1.

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

General Description: 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 16).

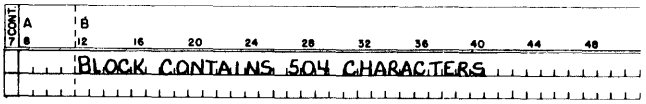


Figure 16. BLOCK CONTAINS

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

General Description: 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 17).

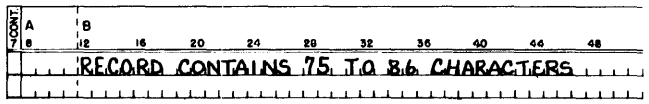


Figure 17. RECORD CONTAINS

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

General Description: This required clause states whether header and trailer tape label records are 1401 COBOL tape labels or omitted. The 1401 COBOL processor can handle only 1401 COBOL tape labels as described previously. For punched-card files the OMITTED option must be used.

Example: Figure 18 shows a LABEL RECORD entry for a punched-card input file.

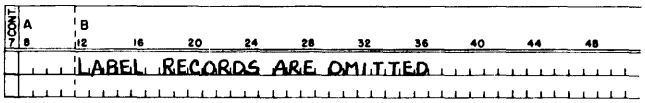


Figure 18. LABEL RECORDS

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:

Card Columns	*Contents*
	YR DAY
1-5	XXXXX
40-46	L005199
47-53	N000000
54-60	N000000
61-67	N000000
68-71	1040

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

General Description: 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 header labels are used.

Data-name-1 and *data-name-2* are the names of the fields contained in the header label record. *Literal-1* and *literal-2* refer to the contents of the respective fields. Figure 19 is a chart showing the various data names and the lengths of their associated literals (AN represents alphanumeric values and N represents numeric values). It also shows the relationship between use of the entries and the type of label checking that will be applied to an input or output file. All entries in the chart, except those noted by one or two asterisks, are required.

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

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

General Description: 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.

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.

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

Record-Description Entry

General Description: The record-description entries in the COBOL source program provide detailed information about each item of data that will be needed during the running of the object program. Each such item must have its own entry consisting of a level-number, a data-name, and a series of independent clauses.

Reference Format

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

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

[OCCURS integer-2 TIME[S]]

[POINT LOCATION IS {LEFT / RIGHT} integer-3 PLACE[S]]

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

[PICTURE IS Any allowable combination of characters and
                symbols as described in Chapter 6 (COBOL GI)]

[JUSTIFIED {LEFT / RIGHT}]

[{ZERO SUPPRESS / CHECK PROTECT / FLOAT DOLLAR SIGN}

                [LEAVING integer-4 PLACE[S]]]

[BLANK WHEN ZERO]

[{VALUE IS / VALUES ARE} literal-1 [THRU literal-2] [literal-3

                [THRU literal-4]...]].
```

General Description: The level-number shows the relationship between items in a record.

The highest level is 01 and the lowest level is 49. Level 77 applies to non-contiguous items of data that are elementary in themselves. Level 88 denotes a condition name and must appear immediately after the entry that describes the data name with which a condition name is associated.

Each level number must be associated with a data-name or the key word FILLER. FILLER must describe items that appear in records but are not referred to within procedure statements.

DATA-NAME	Complete Checking		Partial Checking	
	INPUT	OUTPUT	INPUT	OUTPUT
ID or IDENTIFICATION	10 AN	10 AN	10 AN	10 AN
CREATION DATE	5 N			
RETENTION-CYCLE	3 N	3 N		3 N
FILE-SERIAL-NUMBER	* 5 N	* 5 N		
REEL-SEQUENCE-NUMBER	** 3 N	** 3 N		

* The use of a FILE-SERIAL-NUMBER entry implies full label checking.

** If not specified, 001 will be assumed.

Figure 19. Data-Names and Lengths of Their Associated Literals

Items must be written in the record-description entry in the same order in which they appear in the record.

```
[ SIZE IS [integer-1 TO] integer-2 [ { CHARACTER[S] } ] ]
                                   { DIGIT[S]     }

                    [DEPENDING ON data-name]
```

General Description: This clause tells the processor how many characters (or digits) the data item contains.

This size is interpreted by the 1401 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 22 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.

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

```
[ SIZE IS integer-2 [ { CHARACTER[S] } ] ]
                    { DIGIT[S]     }
```

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

Example: Figure 23 shows a SIZE entry for a fixed-length record whose size is eighty characters.

```
[ OCCURS integer-2 TIME[S] ]
```

General Description: The OCCURS clause describes a sequence of data items of the same format. For example, if a rate table contains ten rates, each made up of five characters, fifty storage positions can be reserved for the rate table by using one OCCURS clause. An individual rate from this rate table can be referred to in the PROCEDURE DIVISION by subscripting the data-name assigned to the rates. The maximum number of positions that can be reserved by an OCCURS clause is 999.

Example: Figure 24 shows how an OCCURS clause for a rate table may be coded.

In the PROCEDURE DIVISION a statement using RATE (2) as a subscripted data-name can refer to the second rate in the rate table (Figure 25).

The OCCURS clause may not be used with an item whose level number is 01, 77, or 88. *Integer-2* must be a positive numerical literal having an integral value greater than zero.

```
[ POINT LOCATION IS { LEFT  } integer-3 PLACE[S] ]
                    { RIGHT }
```

General Description: This clause describes the decimal point location for a number so that the processor can provide for the correct alignment of assumed decimal points during computation. It can be used only with an elementary item. *Integer-3* must be a numerical literal with an integral value.

Example: The POINT clause (Figure 26) causes an assumed decimal point to be located two positions to the left of the units position of the item whose data-name is GROSSPAY (999V99).

Note that the assumed decimal point is not included in the size of the item because it will not actually exist in 1401 core storage at program-execution time.

$$\left[\text{CLASS IS } \left\{\begin{array}{l}\underline{\text{ALPHABETIC}}\\ \underline{\text{NUMERIC}}\\ \underline{\text{ALPHANUMERIC}}\\ \underline{\text{AN}}\end{array}\right\}\right]$$

General Description: This clause tells the COBOL processor whether an item is numerical, alphabetic, or a combination of alphabetic and numerical characters with or without special characters (alphanumeric). It is required only if no PICTURE clause is included in the source program. The CLASS clause may describe both elementary and group items at any level.

The sample problem shows COBOL entries without either class or picture clauses. The 1401 COBOL Processor assumes items not specified by either of these clauses to be alphanumeric.

NUMERIC specifies an item that consists entirely of digits 0-9, and a plus or minus sign if one is necessary.

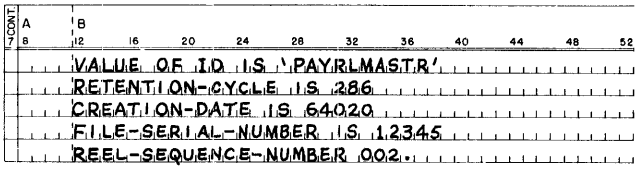


Figure 20. IDENTIFICATION and Retention Cycle

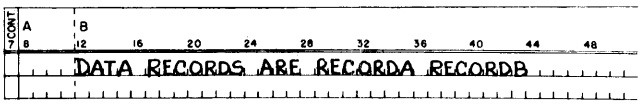


Figure 21. DATA RECORDS

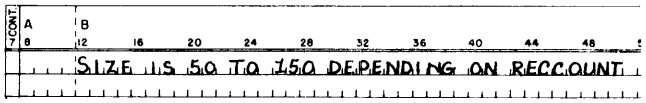


Figure 22. SIZE Variable Length

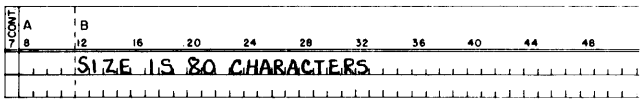


Figure 23. SIZE Fixed Length

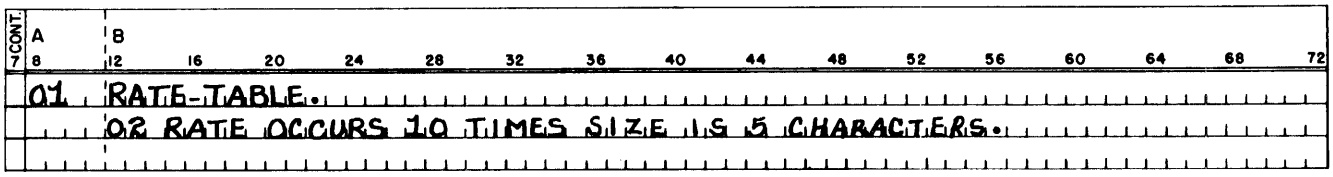


Figure 24. OCCURS

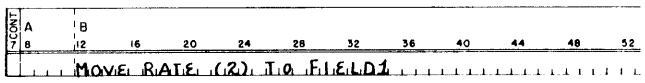


Figure 25. Subscripting

ALPHABETIC specifies an item that consists entirely of the letters of the alphabet and spaces.

ALPHANUMERIC specifies an item that consists of any characters in the 1401 character set in any combination (all numeric, all alphabetic, or mixed).

Example: Figure 27 shows a CLASS clause that defines a numeric field of five digits with a sign over the units position.

[PICTURE IS *Any allowable combination of characters and symbols as described in Chapter 6 (COBOL GI)*]

General Description: The PICTURE clause can describe elementary items. It can be used instead of the SIZE, POINT, CLASS, EDITING, and BLANK clauses of a record-description entry to state the characteristics of an item in a more compact form.

The rules for forming a picture for a data item are given in the COBOL General Information Manual. However, with 1401 COBOL the S symbol for an operational sign should not be used, and the elementary item generated from a PICTURE clause must not exceed 999 positions of core storage.

Example: The PICTURE clause shown in Figure 28 describes an item whose data name is 3C-ONE. The size of this field is five numeric characters with an asumed decimal point two places to the left of the units position.

[JUSTIFIED {LEFT / RIGHT}]

General Description: This clause specifies the position data is to occupy if it is moved during processing from one location to a larger location.

If a JUSTIFIED clause is not included in the COBOL source program, numerical items with be automatically right-justified (the data in the units position of the original location will be placed in the units position of the new location), and the unusued positions to the left will be filled with zeros. If a numerical data item includes an assumed decimal point, alignment is made during the automatic justification.

ALPHABETIC and ALPHANUMERICAL items will be left-justified (the data in the high-order position of the original location will be placed in the high-order position of the new location), and unused positions to the right will be filled with blanks.

The JUSTIFIED clause permits the programmer to reverse the justification of an item (numerical items can be left-justified and alphanumerical items right-justified) except when an assumed decimal point has previously been specified for the item.

Example: The numerical item DEDUCT is eight charac-

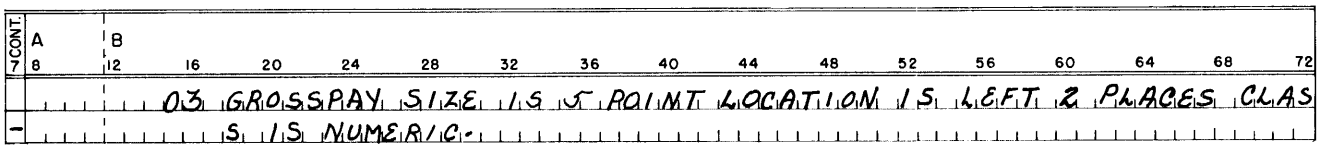


Figure 26. SIZE and POINT Location

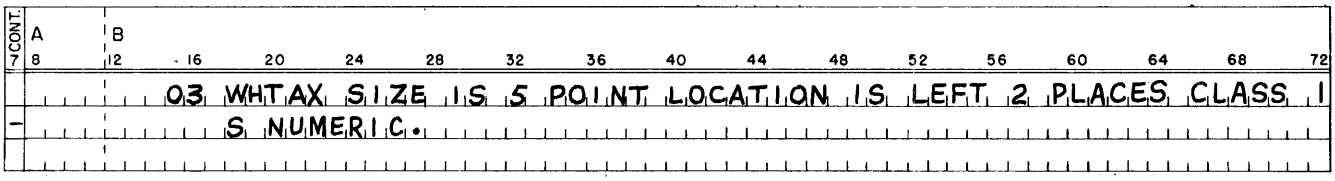


Figure 27. CLASS

ters long. During processing, a six-character field is to be moved to DEDUCT and is to be left-justified. Figure 29 shows a JUSTIFIED clause that specifies that the data moved to the DEDUCT field will be left-justified.

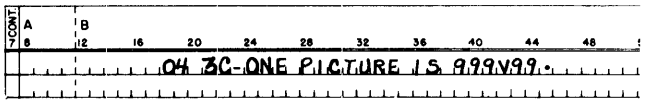


Figure 28. PICTURE

Special Editing Functions

These COBOL editing functions may be used only when the object computer is equipped with the expanded print-edit feature:

1. High-order CR or minus signs and high-order DB or plus signs.
2. Floating plus and minus signs, and floating dollar signs.
3. Check protection (asterisk fill).
4. Decimal suppression for blank or zero fields.

Editing of a single-digit field cannot be specified in editing or PICTURE clauses.

When the editing options for floating plus, minus, and dollar sign are used, more than two floating characters must be specified in the PICTURE or EDITING clause. For example, if $$99 is specified, the $ will not float, but zero suppression will take place. However, if $$$9 is specified, the dollar sign will float and zero suppression will take place.

If a group item is moved to an elementary item that requires editing, the diagnostic INVALID USE OF EDITING will result. (A report item can only receive numeric data; the 1401 tape COBOL compiler considers a group item as alphameric data.)

Editing Clauses

```
[ { ZERO SUPPRESS      }
  { CHECK PROTECT      }
  { FLOAT DOLLAR SIGN  } ]
                [ LEAVING integer-4 PLACE[S] ]
```

General Description: This clause permits the programmer to specify certain kinds of editing without using a PICTURE clause. Only elementary numerical items may be described using the EDITING clause.

ZERO SUPPRESS causes high-order zeros to be replaced with blanks up to but not including the first non-zero digit or an assumed or actual decimal point encountered in a numerical item.

When using zero suppression in either the PICTURE clause or the EDITING clause, the high-order position must not be the only character specifying zero suppression. For example, Z9 is incorrect, but ZZ is correct.

CHECK PROTECT causes all high-order zeros to be replaced with asterisks under the same conditions as ZERO SUPRESS.

FLOAT DOLLAR SIGN causes all high-order zeros to be blanked and a dollar sign to be placed to the left of the first significant character or decimal point. The EDITING clauses can be used only to affect zeros to the left of the decimal point. Zeros to the right of the decimal point must be cleared (if desired) by a BLANK clause.

The LEAVING option restricts the suppression of zeros or insertion of asterisks and the dollar sign by specifying that a certain number of places *(integer-4)* to the left of the decimal point are to be undisturbed (unedited).

Example: Figure 30 shows an EDITING clause that specifies that high-order zeros are to be replaced with blanks. Assume that a field called FICA is to be edited during processing in preparation for printing. If the value of the field moved to FICA appears as 00508 before editing, it will appear as 508 after editing.

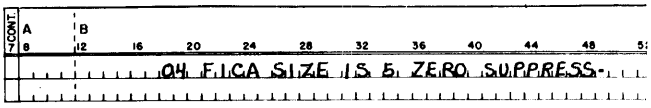


Figure 30. Editing

```
[ BLANK WHEN ZERO ]
```

General Description: This clause causes the described item to be filled with blanks whenever the value (contents) of the item field is zero. It can be used only with elementary items. BLANK WHEN ZERO overrides all editing specifications prescribed by a PICTURE or EDITING clause.

Example: Figure 31 shows a BLANK WHEN ZERO clause used with a FLOAT DOLLAR SIGN clause. Without the BLANK WHEN ZERO clause, a ZERO PAY field would ap-

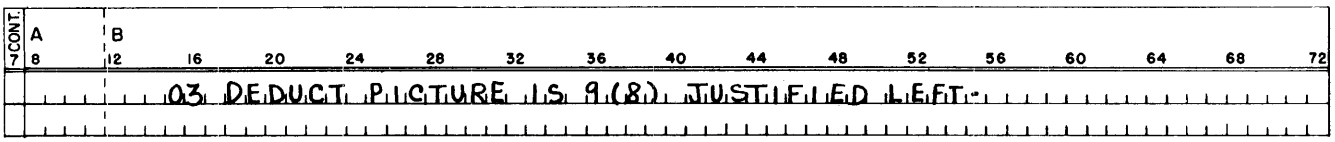


Figure 29. JUSTIFIED

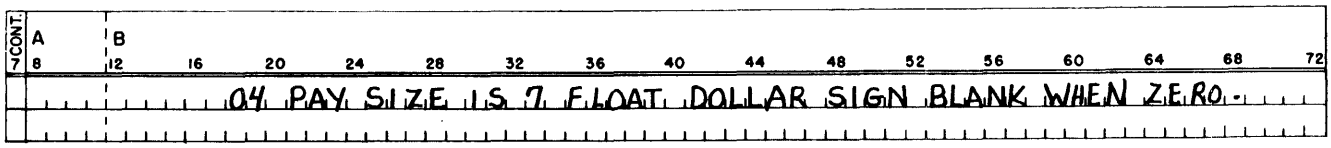


Figure 31. BLANK WHEN ZERO

pear after editing as $.00. The BLANK WHEN ZERO clause causes seven blanks to appear in the PAY field when it is edited for printing.

```
[ { VALUE IS  } literal-1 [THRU literal-2] [ literal-3
  { VALUES ARE }                           [THRU literal-4]... ] ].
```

General Description: 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 with the THRU option 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.

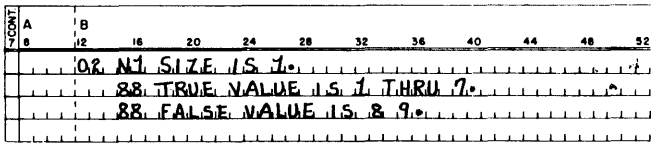


Figure 32. VALUE

The THRU option is not described in the COBOL General Information Manual. It may be used only with condition names as shown in Figure 32.

```
[ REDEFINES data-name-2 ]
```

General Description: This clause permits the programmer to redefine areas of storage that have been previously defined by a record-description entry.

Note the following additional points in connection with the use of REDEFINES:

1. If *data-name-2* is not unique, it must be qualified by one or more additional names until it is unique.
2. *Data-name-2* must not be subscripted.

Example: Figure 33 shows a REDEFINES clause used to define a table of constants.

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 the IBM 1401 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.

Deferred Elements of the Data Division

The COPY option is contained in the COBOL General Information Manual, but is not contained in this version of the IBM 1401 COBOL processor.

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

A group mark should not be declared as an alphameric literal.

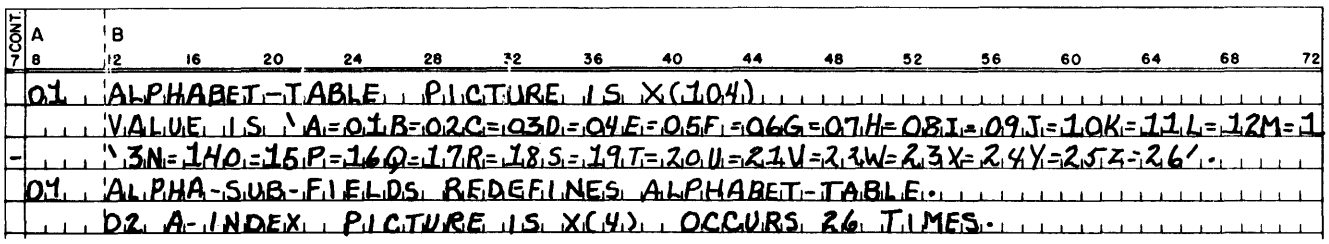


Figure 33. REDEFINES

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 COBOL source program. This additional information is presented in the following section.

The DISPLAY Verb

General Description: The IBM 1403 Printer (1403-P) is the standard output unit for the DISPLAY verb. However, information may also be displayed via the IBM 1402 Card Read-Punch (1402-P). 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 1403 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 channel 1 and 12 (overflow) at object-program execution time.

Examples: The statement shown in Figure 34 will cause the contents of the area whose data name is GRAND-TOTAL to be displayed on the 1403 printer.

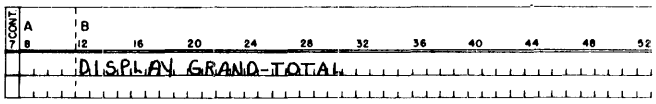


Figure 34. PRINTER DISPLAY

The statement shown in Figure 35 will cause the contents of GRAND-TOTAL to be punched into cards, if the mnemonic-name CARD PUNCH has been assigned to 1402-P in the SPECIAL-NAMES paragraph of the ENVIRONMENT DIVISION.

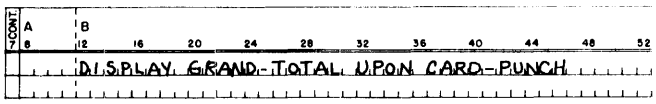


Figure 35. PUNCH DISPLAY

The ACCEPT Verb

General Description: The IBM 1402 Card Read-Punch (1402-R) is the standard input unit for the ACCEPT verb.

Example: Figure 36 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 eighty storage positions are defined by CANCELLATIONS, multiple cards will be read from the 1402 until the area is filled.

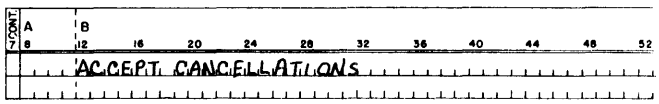


Figure 36. ACCEPT

The WRITE Verb

Reference Format

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

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

General Description: 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. *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 1403 printer (1403-P).

AFTER and BEFORE in the ADVANCING option control printer carriage spacing 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.

Examples: Figures 37, 38, 39, and 40 show sample WRITE statements.

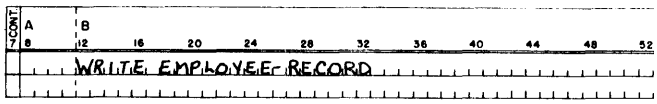


Figure 37. WRITE

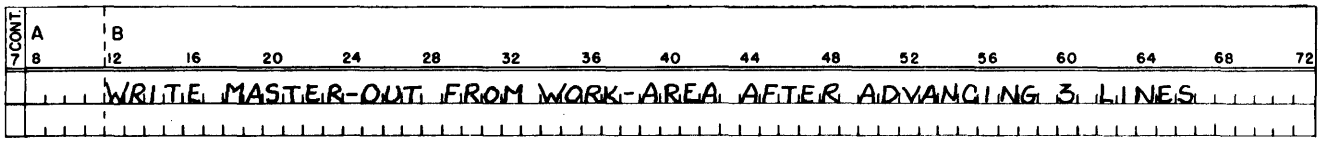


Figure 38. WRITE AND SPACE BEFORE PRINTING

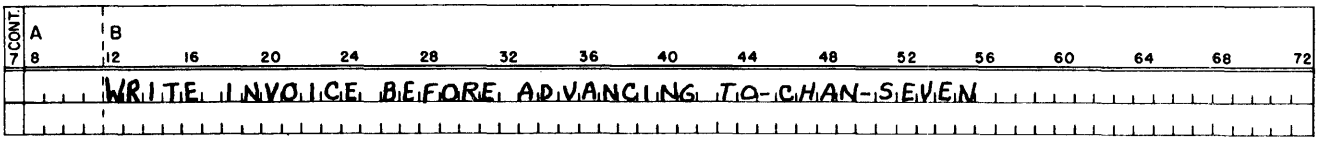


Figure 39. WRITE AND SKIP AFTER PRINTING

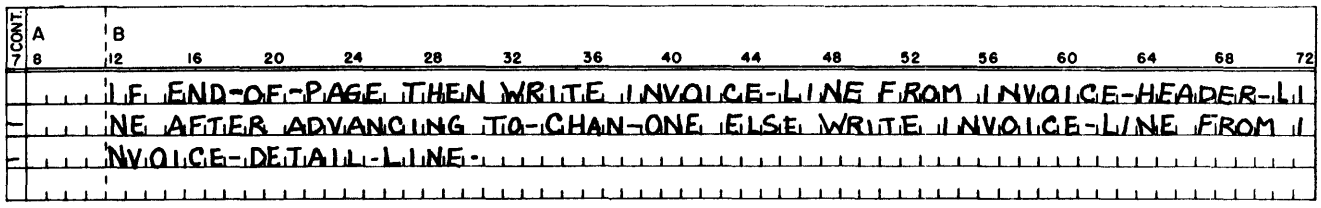


Figure 40. CONDITIONAL WRITE

The EXAMINE Verb

Reference Format

```
EXAMINE data-name { TALLYING { ALL     }                                }
                  {          { LEADING }                                }
                  {          { UNTIL FIRST }                            }
                  {  literal-1 [REPLACING BY literal-2]                 }
                  { REPLACING { ALL     }                               }
                  {           { LEADING }                               }
                  {           { [UNTIL] FIRST }                         }
                  {  literal-3 BY literal-4                             }
```

General Description: The EXAMINE verb is used to replace a given character and/or to count the number of times it appears in a data item.

Any literal used in an EXAMINE statement must be a member of the character set associated with the CLASS specified for *data-name*. Thus, if the description of *data-name* in the DATA DIVISION specifies a CLASS that uses less than the full character set (NUMERIC or ALPHABETIC), then each literal used in an EXAMINE statement must be one of the characters in the restricted set. Thus, if the class of *data-name* is NUMERIC, each literal used in the statement must be a numeric character.

All literals in EXAMINE statements are considered alphanumeric, are one character in length, and are enclosed by quotation marks. When an EXAMINE statement is executed, the examination begins with the leftmost character of the data item and proceeds to the right. Each character in the item represented by the *data-name* is examined in turn. If the data item being examined is numeric, any operational sign associated with the item will be ignored.

The effect of an EXAMINE statement depends on the options employed by the programmer as follows:

If TALLYING is specified:

A count of the number of certain characters in *data-name* is made when the TALLYING option is used. This count replaces the value of a special register called TALLY, which is accessible to the programmer. The count depends on which of three options of TALLYING is used:

1. If ALL is specified, all occurrences of *literal-1* in the data item are counted.
2. If LEADING is specified, the count represents the number of times *literal-1* occurs before a character other than *literal-1* is encountered.
3. If UNTIL FIRST is specified, the count represents the number of characters that are encountered before *literal-1* first occurs.

If REPLACING is specified:

The replacement of characters depends on which of the four options of REPLACING is used when the

REPLACING option is used either with or without the TALLYING option:

1. If ALL is specified, *literal-2* or *literal-4* is substituted each time *literal-1* or *literal-3* occurs. *Literal-2* is substituted for *literal-1*, and *literal-4* is substituted for *literal-3*.
2. If LEADING is specified, the substitution ends when a character other than the literal *(literal-1* or *literal-3)* is encountered or when the rightmost character of the data item is reached.
3. If UNTIL FIRST is specified, the count represents the number of characters that are encountered before *literal-1* first occurs.
4. If FIRST is specified, *literal-3* is replaced by *literal-4* only the first time *literal-3* occurs.

Example: Figure 41 shows a use of the EXAMINE verb.

The ENTER Verb

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

The language name used with 1401 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. 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 the 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. The symbols used in Autocoder statements must be five characters long.

IBM COBOL PROGRAM SHEET Form No. X28-1464 Printed in U.S.A.

PAGE 1 3 | PROGRAM EXAMINE SAMPLE | SYSTEM | SHEET OF
PROGRAMMER | DATE | IDENT 73 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
001		IDENTIFICATION DIVISION.
002		PROGRAM-ID. THE-EXAMINE-VERB.
003		ENVIRONMENT DIVISION.
004		CONFIGURATION SECTION.
005		SOURCE-COMPUTER. IBM-1401 MEMORY SIZE 4000 CHARACTERS.
006		OBJECT-COMPUTER. IBM-1401 MEMORY SIZE 4000 CHARACTERS.
007		DATA DIVISION.
008		WORKING-STORAGE SECTION.
009		01 ANSWER PICTURE IS XXXXXX.
010		CONSTANT SECTION.
011		77 CONST1 PICTURE IS XXXXXX VALUE IS '101010'.
012		77 CONST2 PICTURE IS X(6) VALUE IS '111333'.
013		77 CONST3 PICTURE IS X(6) VALUE IS '225522'.
014		PROCEDURE DIVISION.
015		START. MOVE CONST1 TO ANSWER.
016		EXAMINE ANSWER TALLYING ALL '1' REPLACING BY '0'.
017		DISPLAY ANSWER.
018		EXAMINE ANSWER REPLACING ALL '0' BY 'X'.
019		DISPLAY ANSWER.
020		MOVE CONST2 TO ANSWER.
021		EXAMINE ANSWER REPLACING LEADING '1' BY ' '. DISPLAY ANSWER.
022		MOVE CONST3 TO ANSWER.
023		EXAMINE ANSWER TALLYING UNTIL FIRST '3'. DISPLAY TALLY.
024		STOP 'END OF JOB'.

Figure 41. EXAMINE Verb

3. Autocoder statements can be written to refer to COBOL names if they are related by entries in the SPECIAL-NAMES section of the COBOL program. However, COBOL statements cannot be written to refer to Autocoder names.

4. 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.

5. No 1401 SPS statements can be included.

Example: Figure 42 is an example that includes a section of Autocoder statements.

6. Macro instructions may be given which refer to macros in the Autocoder library.

The STOP Verb

Reference Format

STOP { RUN / *literal* }

General Description: This statement produces a 1401 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 is numeric and within the range 0-99, the literal 000-099 is displayed in the B-register if the halt occurs during the running of the object program.

IBM

COBOL PROGRAM SHEET

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

PAGE 1 3 | PROGRAM ENTER SAMPLE | SYSTEM 1 | SHEET 1 OF 3
001 | PROGRAMMER | DATE | IDENT. 73 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
010		IDENTIFICATION DIVISION.
020		PROGRAM-ID. THE-ENTER-VERB.
030		ENVIRONMENT DIVISION.
040		CONFIGURATION SECTION.
050		SOURCE-COMPUTER. IBM 1401 MEMORY SIZE IS 4000 CHARACTERS.
060		OBJECT-COMPUTER. IBM 1401 MEMORY SIZE IS 4000 CHARACTERS.
070		SPECIAL NAMES.
080		AREA1 IS ANSWER.
090		MCAND IS CONST1.
100		MPLYR IS CONST2.
110		START IS START.
120		ANSBK IS ANSWER-BUCKET.
130		DATA DIVISION.
140		WORKING-STORAGE SECTION.
150		01 ANSWER CLASS IS NUMERIC SIZE IS 30.
160		01 PERCENTAGE CLASS IS NUMERIC SIZE IS 10.
170		01 ANSWER-BUCKET NUMERIC SIZE IS 8.
180		CONSTANT SECTION.
190		77 CONST1 PICTURE IS 99999 VALUE IS +42800.
200		77 CONST2 PICTURE IS 99 VALUE IS +18.
210		PROCEDURE DIVISION.
220		BEGIN. ENTER AUTOCODER.

Figure 42. ENTER Verb, Part 1 of 3

IBM

Program ENTER SAMPLE

Programmed by

Date

INTERNATIONAL BUSINESS MACHINES CORPORATION

IBM 1401 AND 1410 DATA PROCESSING SYSTEMS

AUTOCODER CODING SHEET

Form X24-1350-1 Printed in U.S.A.

Identification

Page No. 02 of 03

Line	Label	Operation	OPERAND
0,1,0		MLCWA	ZEROS, AREA1
0,2,0		MLCWA	MPLYR, AREA1-20
0,3,0	ZRTST.	BCE	TSTWM, AREA1-20, 0
0,4,0		BCE	TSTWM, AREA1-20, 8
0,5,0		A	MCAND, AREA1-8
0,6,0		S	'1', AREA1-20
0,7,0		B	ZRTST.
0,8,0	TSTWM	BWZ	LAST1, AREA1-20, 1
0,9,0		MLCWA	AREA1-1, AREA1
1,0,0		B	ZRTST.
1,1,0	LAST1.	BWZ	LAST3, AREA1-12, 1
1,2,0		SW	AREA1-12
1,3,0		MLCWA	AREA1-7, ANSBK
1,4,0		MLC	'1', SWBCK
1,5,0		BCE	LAST3, SWBCK
1,6,0		CW	AREA1-12
1,7,0	LAST2.	B	START.
1,8,0	LAST3.	MLC	'1', SWBCK
1,9,0		B	LAST2
2,0,0	ZEROS.	DCW	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
2,1,0	SWBCK	DCW	0
2,2,0			
2,3,0			
2,4,0			
2,5,0			

Figure 42. ENTER Verb, Part 2 of 3

IBM

COBOL PROGRAM SHEET

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

PAGE 003 | PROGRAM ENTER SAMPLE | SYSTEM | SHEET 3 OF 3

PROGRAMMER | DATE | IDENT.

SERIAL	CONT.	A B
0,1,0		START. ENTER COBOL.
0,2,0		DISPLAY ANSWER-BUCKET.
0,3,0		IF ANSWER-BUCKET = 3000 OR GREATER THAN 5900 GO TO
0,4,0		COMPUTE-ONE OTHERWISE GO TO NEXT-TRANSACTION.
0,5,0		COMPUTE-ONE. COMPUTE PERCENTAGE ROUNDED = ANSWER-BUCKET * .05.
0,6,0		DISPLAY PERCENTAGE.
0,7,0		NEXT-TRANSACTION. DISPLAY 'END OF ENTER EXAMPLE'.
0,8,0		STOP 'END OF EXAMPLE'.

Figure 42. ENTER Verb, Part 3 of 3

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 when an object-program halt occurs.

Example: Figure 43 shows the STOP statement.

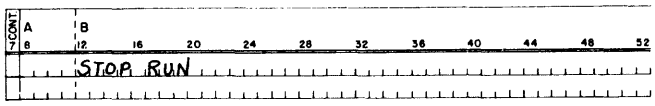


Figure 43. STOP Verb

The OPEN and CLOSE Verbs

The COBOL language, as specified in the COBOL General Information Manual, provides the ability to open an output file, process it, close it, and subsequently open it as an input file. It also provides for opening an input file, processing it, closing it, and subsequently opening it as an output file. These procedures are not handled by this version of the 1401 COBOL processor, and are therefore classified as deferred elements.

Exponents

IBM 1401 COBOL provides for integer or non-integer powers to be used in writing exponents. The sign of the power can be either plus or minus. 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-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* }

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.

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 44 shows outlines for four simple conditional statements. Figure 45 shows an outline for one nested conditional IF statement that produces the same results as the four simple conditional statements shown in Figure 44.

Figure 46 shows an excerpt from a COBOL program in which four simple relational conditional expressions are substituted for the conditions shown in Figures 44 and 45.

The block diagram in Figure 47 shows the logic flow of the nested IF statement in Figure 46.

Added Elective Elements of the Procedure Division

The ADVANCING option of the WRITE verb is not contained in the COBOL General Information Manual, but is contained in the 1401 COBOL processor. Conditional statements within conditional statements are permitted.

```
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 44. 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 45. Nested Conditional IF Statements

IBM COBOL PROGRAM SHEET

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

PAGE 1 3 | PROGRAM | SYSTEM | SHEET OF
PROGRAMMER | DATE | IDENT. 73 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

```
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 2 TO ANSWER. GO TO START.

LABEL-2. MOVE 4 TO ANSWER. GO TO START.

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

LABEL-4. SUBTRACT 2 FROM 4 GIVING ANSWER. GO TO START.
```

Figure 46. Nested Program Sample for Conditional IF Statements

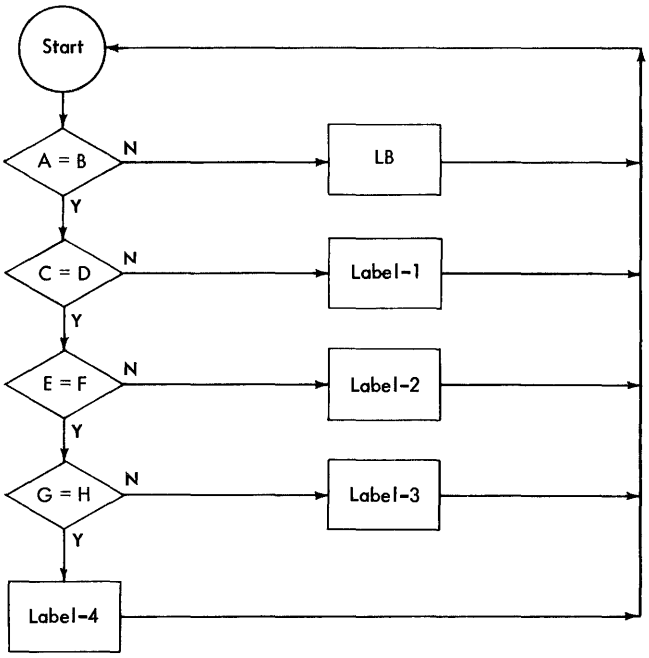


Figure 47. Conditional Logic

Deferred Elements of the Procedure Division

These elements are described in the COBOL General Information Manual but are not implemented by this version of the 1401 COBOL processor:

1. The REEL option of the CLOSE verb.
2. The CORRESPONDING option of the MOVE verb (elective).
3. The ability to process a given file as both an input file and an output file in the same program.
4. The ability to use a group mark as an alphameric literal.
5. The ability to use quote signs (@) within a NOTE statement.

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 IBM 1401 character set may be used only for alphanumeric literals. The following are COBOL (Set H) special characters with their equivalents in the IBM 1401 character set:

Card Code	COBOL (SET H)	1401	*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.

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 publication.

ADVANCING
BEFORE
LINES
VALUES
ID
RETENTION-CYCLE
TAPE
TAPES
NO-RELEASE
NO-OVERLAP
NO-PRINT-STORAGE
1402-R
1402-P
1403-P
1403-CT
1403-P-CB
1403-P-C9
1403-P-CV
1401-SS
CREATION-DATE
FILE-SERIAL-NUMBER
REEL-SEQUENCE-NUMBER

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.

Example: 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 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 symbol (−) must appear in column7, and a quotation mark must appear in column 12. If the last character of an alphanumeric literal appears in column 72, column 7 must contain a continuation mark, and columns 12 and 13 *must both* contain quotation marks.

Sample Problem

Here is a sample problem that is representative of file maintenance applications. It is not a source program for a unique problem.

The IDENTIFICATION, ENVIRONMENT, and DATA DIVISIONS are complete in themselves. The PROCEDURE DIVISION contains only one statement that relates to device- and switch-name entries in the ENVIRONMENT DIVISION, and illustrates the ADVANCING option of the WRITE verb.

Figures 48, 49, 50, 51, and 52 describe the 1401 configuration and input and output record formats for the problem. Figure 48 shows the configuration for the object-1401 system. Figure 49 shows the master input and output card record formats. Figure 50 is the master-record block format for the input and output tapes. Figure 51 is the new master-card record format. Figure 52 is the form layout for the invoice. Figure 53 is the sample COBOL program.

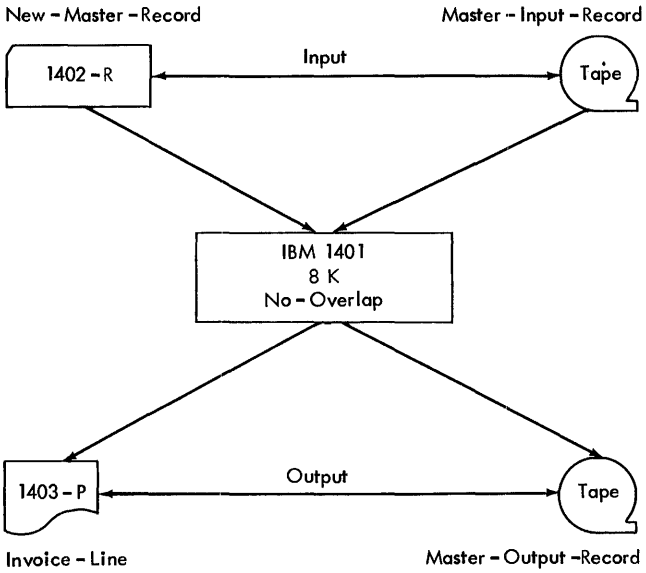


Figure 48. IBM 1401 Object Machine Configuration

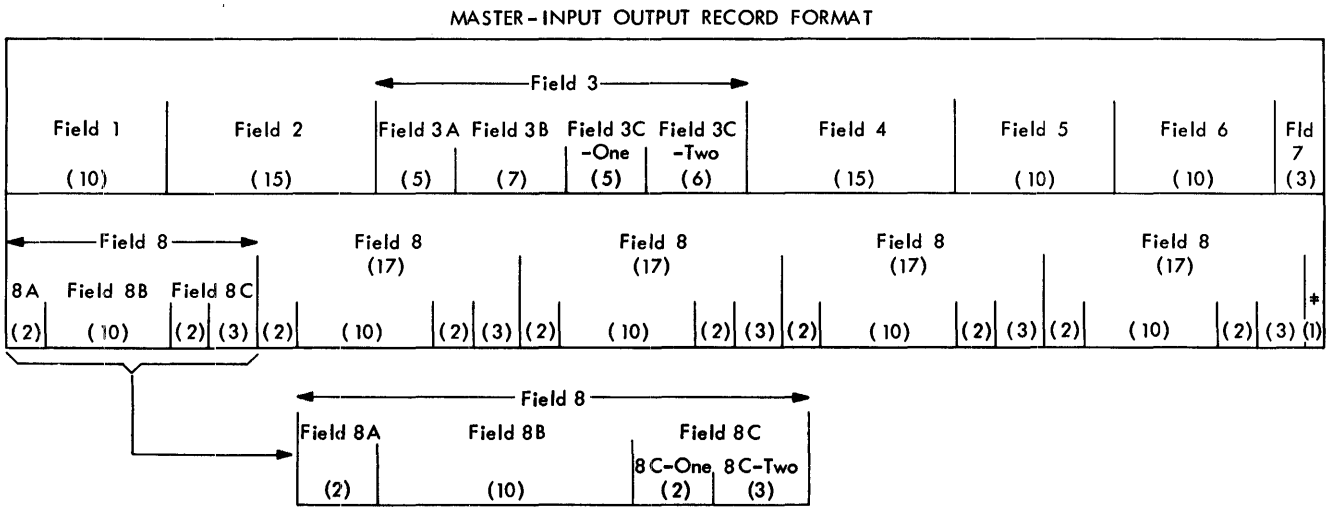


Figure 49. Master Input and Output Record Format

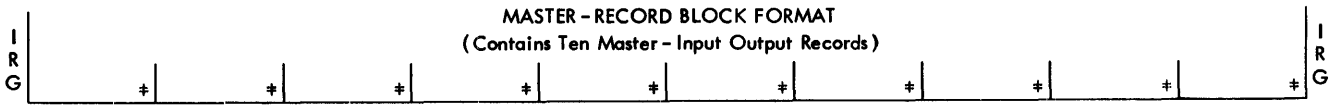


Figure 50. Master Record Block Format

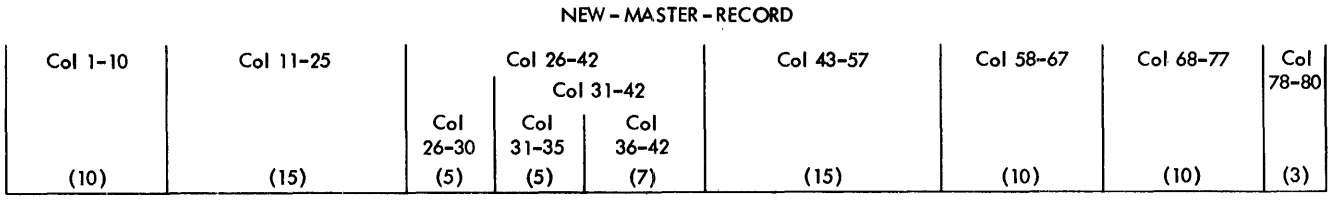


Figure 51. New Master Card Format

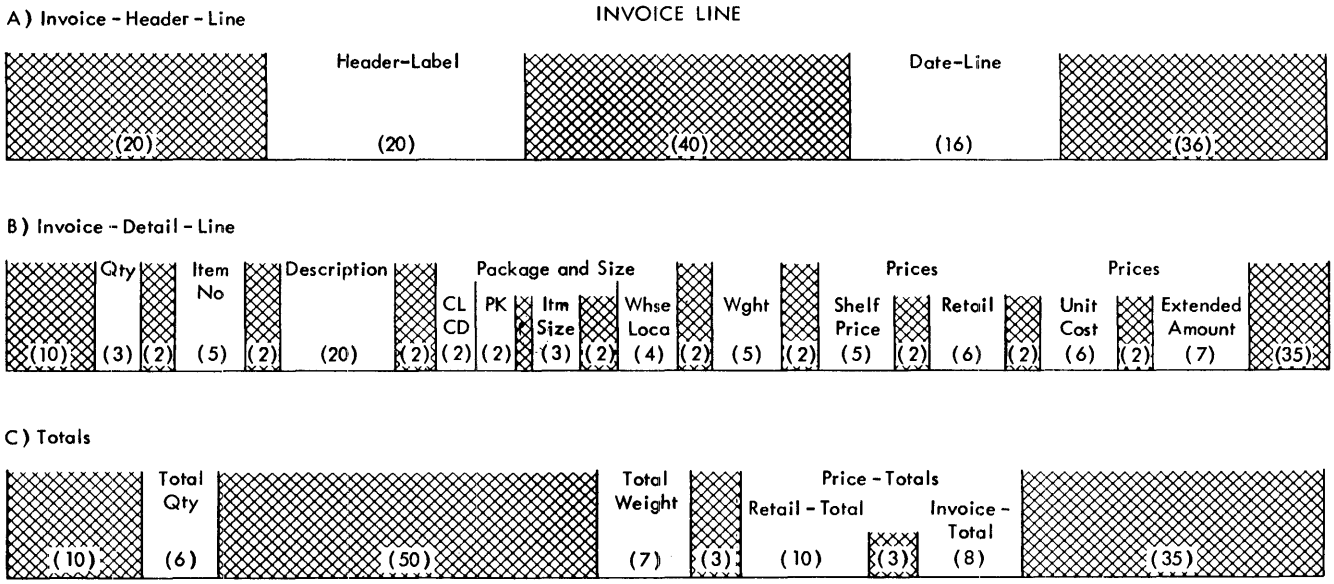
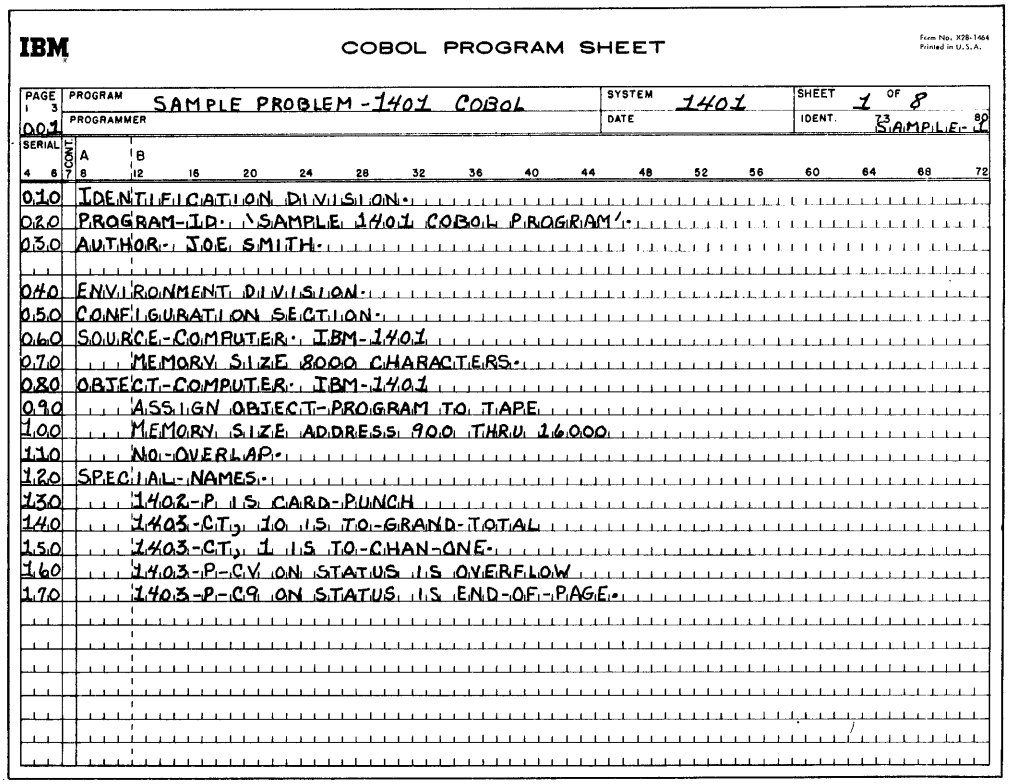


Figure 52. Invoice Form Layout



IBM

COBOL PROGRAM SHEET

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

PAGE 1 3 | PROGRAM SAMPLE PROBLEM -1401 COBOL | SYSTEM 1401 | SHEET 1 OF 8

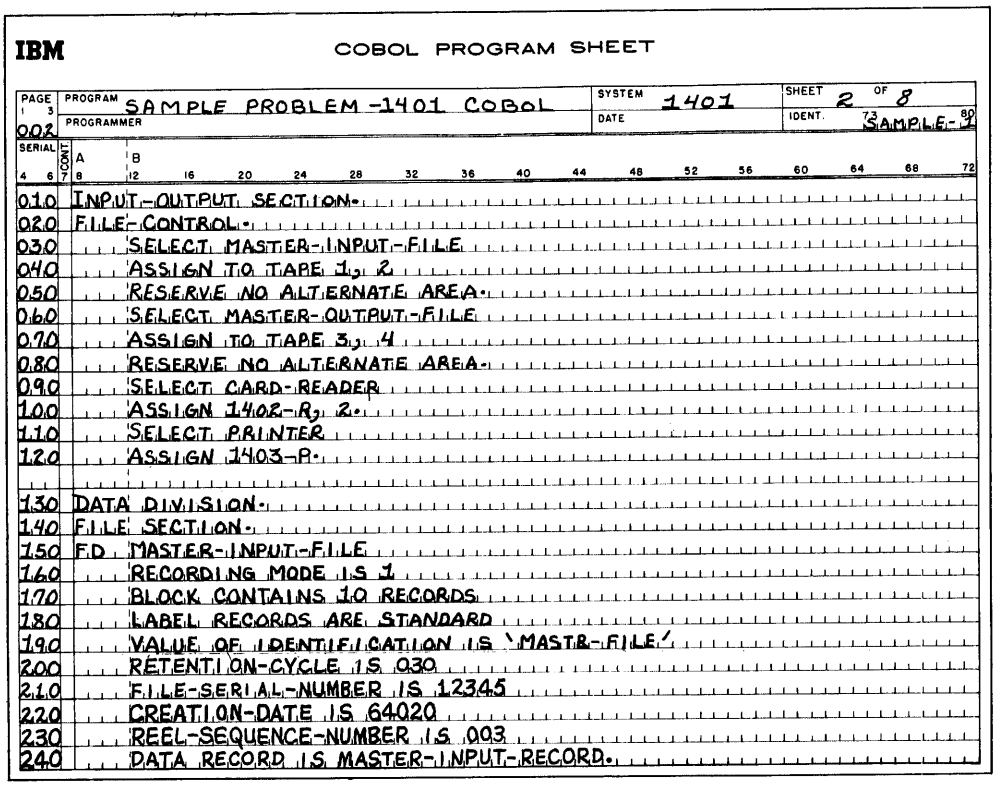
001 | PROGRAMMER | DATE | IDENT. 73 SAMPLE-1 80

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

```
010 IDENTIFICATION DIVISION.
020 PROGRAM-ID. 'SAMPLE 1401 COBOL PROGRAM'.
030 AUTHOR. JOE SMITH.

040 ENVIRONMENT DIVISION.
050 CONFIGURATION SECTION.
060 SOURCE-COMPUTER. IBM-1401
070     MEMORY SIZE 8000 CHARACTERS.
080 OBJECT-COMPUTER. IBM-1401
090     ASSIGN OBJECT-PROGRAM TO TAPE
100     MEMORY SIZE ADDRESS 900 THRU 16000
110     NO-OVERLAP.
120 SPECIAL-NAMES.
130     1402-P IS CARD-PUNCH
140     1403-CT, 10 IS TO-GRAND-TOTAL
150     1403-CT, 1 IS TO-CHAN-ONE.
160     1403-P-C7 ON STATUS IS OVERFLOW
170     1403-P-C9 ON STATUS IS END-OF-PAGE.
```

Figure 53. Sample COBOL Program, Part 1 of 8



IBM

COBOL PROGRAM SHEET

PAGE 1 3 | PROGRAM SAMPLE PROBLEM -1401 COBOL | SYSTEM 1401 | SHEET 2 OF 8

002 | PROGRAMMER | DATE | IDENT. 73 SAMPLE-2 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

```
010 INPUT-OUTPUT SECTION.
020 FILE-CONTROL.
030     SELECT MASTER-INPUT-FILE
040     ASSIGN TO TAPE 1, 2
050     RESERVE NO ALTERNATE AREA.
060     SELECT MASTER-OUTPUT-FILE
070     ASSIGN TO TAPE 3, 4
080     RESERVE NO ALTERNATE AREA.
090     SELECT CARD-READER
100     ASSIGN 1402-R, 2.
110     SELECT PRINTER
120     ASSIGN 1403-R.

130 DATA DIVISION.
140 FILE SECTION.
150 FD  MASTER-INPUT-FILE
160     RECORDING MODE IS 1
170     BLOCK CONTAINS 10 RECORDS
180     LABEL RECORDS ARE STANDARD
190     VALUE OF IDENTIFICATION IS 'MASTR-FILE'
200     RETENTION-CYCLE IS 030
210     FILE-SERIAL-NUMBER IS 12345
220     CREATION-DATE IS 64020
230     REEL-SEQUENCE-NUMBER IS 003
240     DATA RECORD IS MASTER-INPUT-RECORD.
```

Figure 53. Sample COBOL Program, Part 2 of 8

IBM

COBOL PROGRAM SHEET

PAGE 003 | PROGRAM SAMPLE PROBLEM - 1401 COBOL | PROGRAMMER | SYSTEM 1401 | DATE | SHEET 3 OF 8 | IDENT. SAMPLE-1

SERIAL 4 6	CONT. 7	A 8	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
010		01	MASTER-INPUT-RECORD.
020			02 FIELD1 PICTURE IS A(10).
030			02 FIELD2 PICTURE IS A(15).
040			02 FIELD3.
050			03 FIELD-3A PICTURE IS X(5).
060			03 FIELD-3B PICTURE IS X(7).
070			03 FIELD-3C.
080			04 FIELD3C-ONE PICTURE IS 999V99.
090			04 FIELD3C-TWO PICTURE IS 99999V9.
100			02 FIELD4 PICTURE IS A(15).
110			02 FIELD5 PICTURE IS X(10).
120			02 FIELD6 PICTURE IS 9(8)V99.
130			02 FIELD7 PICTURE IS 999.
140			88 TYPE1 VALUE IS 1 THRU 157, 175 THRU 250.
150			88 TYPE2 VALUE IS 359 THRU 750.
160			88 TYPE3 VALUE IS 751 THRU 999.
170			02 FIELD8 OCCURS 5 TIMES.
180			03 FIELD-8A PICTURE IS 99.
190			03 FIELD-8B PICTURE IS 9(10).
200			03 FIELD-8C.
210			04 FIELD8C-ONE PICTURE IS 99.
220			04 FIELD8C-TWO PICTURE IS 999.
230			02 RECORD-MARK SIZE IS 1.

Figure 53. Sample COBOL Program, Part 3 of 8

IBM

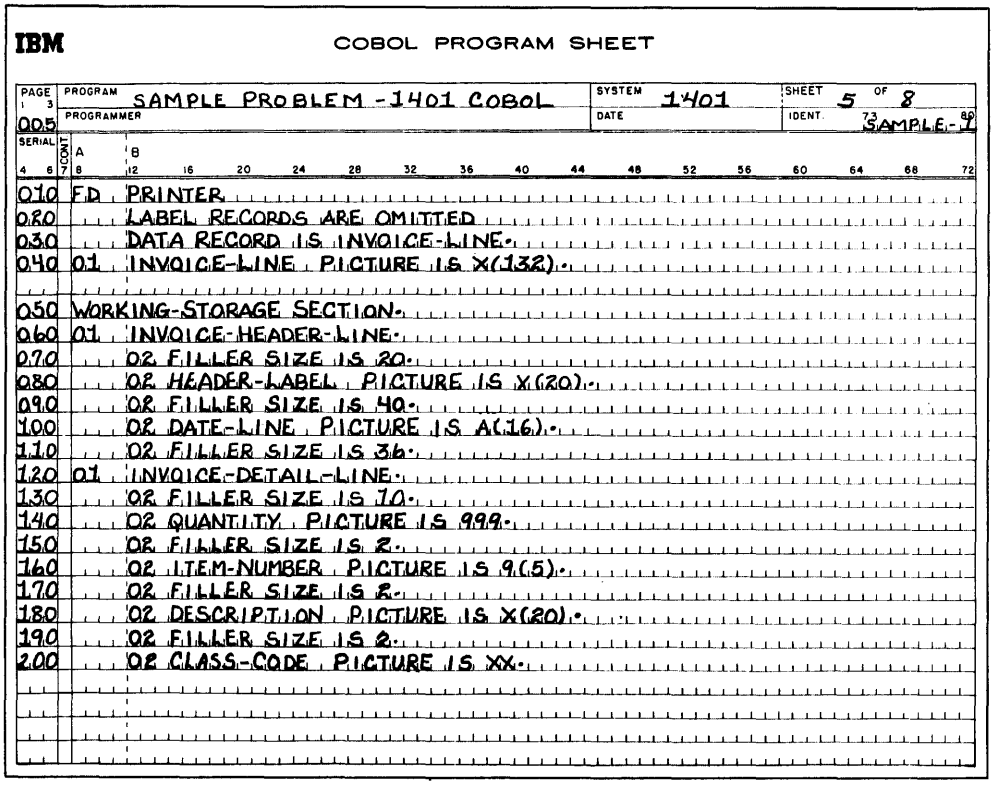
COBOL PROGRAM SHEET

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

PAGE 004 | PROGRAM SAMPLE PROBLEM-1401 COBOL | PROGRAMMER | SYSTEM 1401 | DATE | SHEET 4 OF 8 | IDENT. SAMPLE-2

SERIAL 4 6	CONT. 7	A 8	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
010		FD	MASTER-OUTPUT-FILE
020			RECORDING MODE IS 1
030			BLOCK CONTAINS 10 RECORDS
040			LABEL RECORDS ARE STANDARD
050			VALUE OF ID IS 'MASTR-FILE'
060			RETENTION-CYCLE IS 090
070			DATA RECORD IS MASTER-OUTPUT-RECORD.
080		01	MASTER-OUTPUT-RECORD
090			SIZE IS 172 AN CHARACTERS.
100		FD	CARD-READER
110			RECORDING MODE IS 1
120			LABEL RECORDS ARE OMITTED
130			DATA RECORD IS NEW-MASTER-RECORD.
140		01	NEW-MASTER-RECORD.
150			02 COL1-10 PICTURE IS A(10).
160			02 COL11-25 PICTURE IS A(15).
170			02 COL26-42.
175			03 COL26-30 PICTURE IS X(5).
180			03 COL31-42.
190			04 COL31-35 PICTURE IS 999V99.
200			04 COL36-42 PICTURE IS 9(6)V9.
210			02 COL43-57 PICTURE IS A(15).
220			02 COL58-67 PICTURE IS X(10).
230			02 COL68-77 PICTURE IS 9(8)V99.
240			02 COL78-80 PICTURE IS 999.

Figure 53. Sample COBOL Program, Part 4 of 8



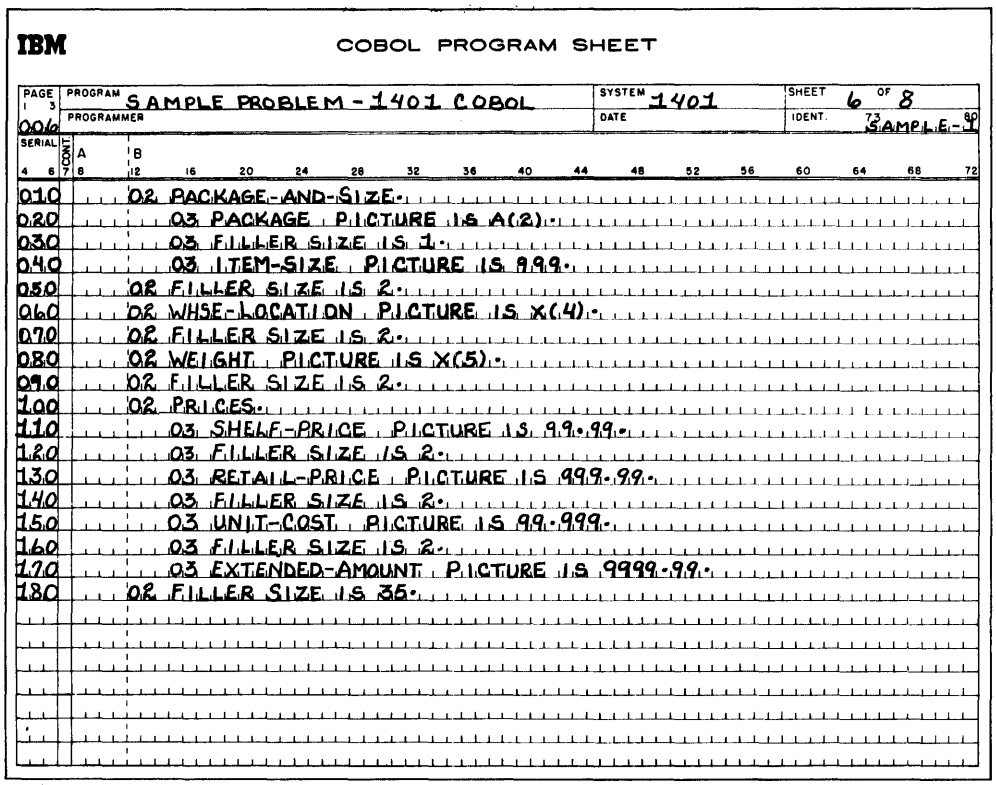
IBM COBOL PROGRAM SHEET

PAGE 3 | PROGRAM SAMPLE PROBLEM - 1401 COBOL | SYSTEM 1401 | SHEET 5 OF 8
005 | PROGRAMMER | DATE | IDENT. SAMPLE-1

```
SERIAL  CONT  A   B
4  6    7    8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
010  FD  PRINTER
020      LABEL RECORDS ARE OMITTED
030      DATA RECORD IS INVOICE-LINE.
040  01  INVOICE-LINE PICTURE IS X(132).

050  WORKING-STORAGE SECTION.
060  01  INVOICE-HEADER-LINE.
070      02 FILLER SIZE IS 20.
080      02 HEADER-LABEL PICTURE IS X(20).
090      02 FILLER SIZE IS 40.
100      02 DATE-LINE PICTURE IS A(16).
110      02 FILLER SIZE IS 36.
120  01  INVOICE-DETAIL-LINE.
130      02 FILLER SIZE IS 10.
140      02 QUANTITY PICTURE IS 999.
150      02 FILLER SIZE IS 2.
160      02 ITEM-NUMBER PICTURE IS 9(5).
170      02 FILLER SIZE IS 2.
180      02 DESCRIPTION PICTURE IS X(20).
190      02 FILLER SIZE IS 2.
200      02 CLASS-CODE PICTURE IS XX.
```

Figure 53. Sample COBOL Program, Part 5 of 8

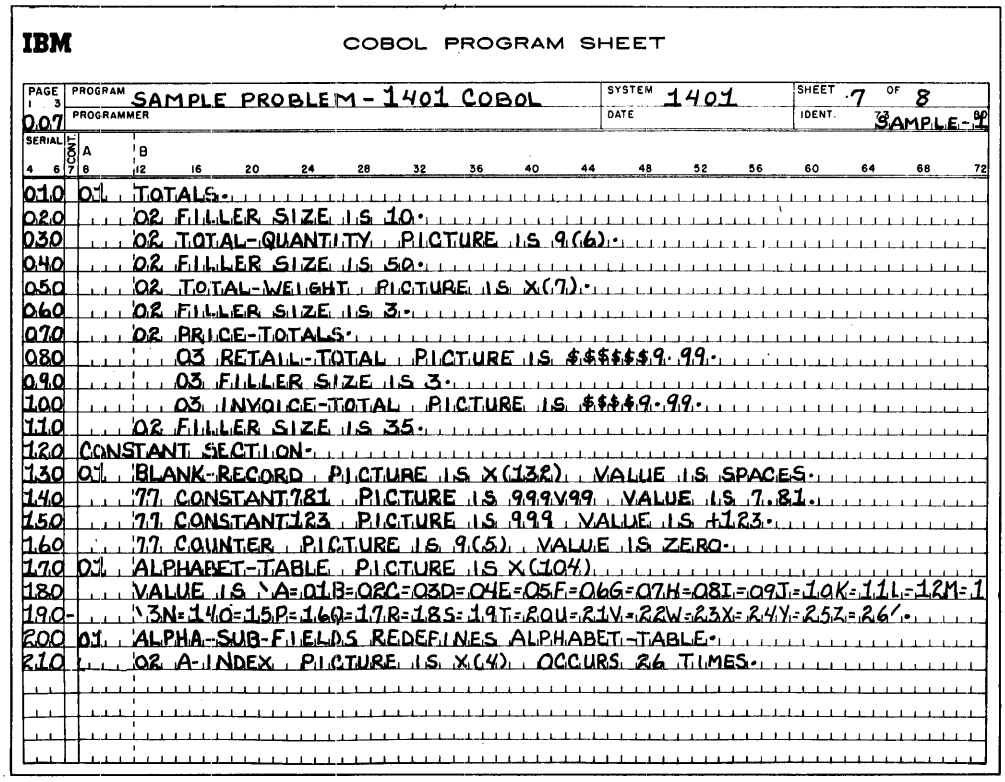


IBM COBOL PROGRAM SHEET

PAGE 3 | PROGRAM SAMPLE PROBLEM - 1401 COBOL | SYSTEM 1401 | SHEET 6 OF 8
006 | PROGRAMMER | DATE | IDENT. SAMPLE-1

```
SERIAL  CONT  A   B
4  6    7    8   12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
010      02 PACKAGE-AND-SIZE.
020         03 PACKAGE PICTURE IS A(2).
030         03 FILLER SIZE IS 1.
040         03 ITEM-SIZE PICTURE IS 999.
050      02 FILLER SIZE IS 2.
060      02 WHSE-LOCATION PICTURE IS X(4).
070      02 FILLER SIZE IS 2.
080      02 WEIGHT PICTURE IS X(5).
090      02 FILLER SIZE IS 2.
100      02 PRICES.
110         03 SHELF-PRICE PICTURE IS 99.99.
120         03 FILLER SIZE IS 2.
130         03 RETAIL-PRICE PICTURE IS 999.99.
140         03 FILLER SIZE IS 2.
150         03 UNIT-COST PICTURE IS 99.999.
160         03 FILLER SIZE IS 2.
170         03 EXTENDED-AMOUNT PICTURE IS 9999.99.
180      02 FILLER SIZE IS 35.
```

Figure 53. Sample COBOL Program, Part 6 of 8

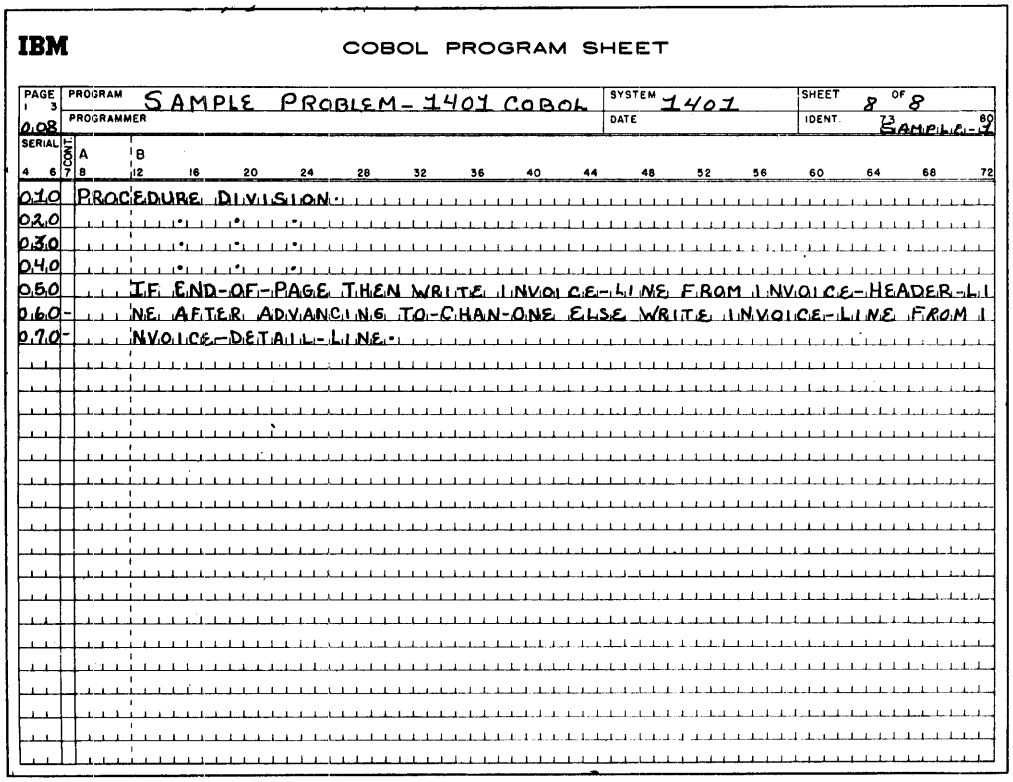


IBM COBOL PROGRAM SHEET

PROGRAM: SAMPLE PROBLEM - 1401 COBOL | SYSTEM: 1401 | SHEET: 7 OF 8
PROGRAMMER | DATE | IDENT: SAMPLE-1

```
0.07
SERIAL  CONT  A  B
4 6     7     8  12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
010     01    TOTALS.
020           02 FILLER SIZE IS 10.
030           02 TOTAL-QUANTITY PICTURE IS 9(6).
040           02 FILLER SIZE IS 50.
050           02 TOTAL-WEIGHT PICTURE IS X(7).
060           02 FILLER SIZE IS 3.
070           02 PRICE-TOTALS.
080              03 RETAIL-TOTAL PICTURE IS $$$$$$9.99.
090              03 FILLER SIZE IS 3.
100              03 INVOICE-TOTAL PICTURE IS $$$$9.99.
110           02 FILLER SIZE IS 35.
120  CONSTANT SECTION.
130  01  BLANK-RECORD PICTURE IS X(132) VALUE IS SPACES.
140      77 CONSTANT781 PICTURE IS 999V99 VALUE IS 7.81.
150      77 CONSTANT123 PICTURE IS 999 VALUE IS +123.
160      77 COUNTER PICTURE IS 9(5) VALUE IS ZERO.
170  01  ALPHABET-TABLE PICTURE IS X(104)
180      VALUE IS \A=01B=02C=03D=04E=05F=06G=07H=08I=09J=10K=11L=12M=1
190-     \3N=140=15P=160=17R=185=19T=20U=21V=22W=23X=24Y=25Z=26'.
200  01  ALPHA-SUB-FIELDS REDEFINES ALPHABET-TABLE.
210      02 A-INDEX PICTURE IS X(4) OCCURS 26 TIMES.
```

Figure 53. Sample COBOL Program, Part 7 of 8



IBM COBOL PROGRAM SHEET

PROGRAM: SAMPLE PROBLEM - 1401 COBOL | SYSTEM: 1401 | SHEET: 8 OF 8
PROGRAMMER | DATE | IDENT: SAMPLE-1

```
0.08
SERIAL  CONT  A  B
4 6     7     8  12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
010  PROCEDURE DIVISION.
020
030
040
050      IF END-OF-PAGE THEN WRITE INVOICE-LINE FROM INVOICE-HEADER-LI
060-     NE AFTER ADVANCING TO-CHAN-ONE ELSE WRITE INVOICE-LINE FROM I
070-     NVOICE-DETAIL-LINE.
```

Figure 53. Sample COBOL Program, Part 8 of 8

Notes

Addition Notes

When using the ADD verb (or when using a COMPUTE statement involving an add operation), the data-names being summed must be placed in order of ascending decimal size in the statement. The smallest decimal field must be first followed by an equal or larger decimal field.

Division Notes

In order to ensure correct decimal alignment when using the DIVIDE verb with the GIVING option (or when using a COMPUTE statement involving a divide operation), 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. Also, the ROUNDED option will have no effect unless this rule is followed.

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 1401 COBOL-generated program. Following the suggestions are two programs. The original program (Figure 54) requires approximately 2,800 positions of core storage. By applying a few of the suggestions to the second program (Figure 55) the core storage requirement is reduced to approximately 1,900 positions of core storage, representing a saving of 33 percent.

The changed statements utilize redefinition, equal decimal alignment, alphabetic compare, and the deletion of a subroutine caused by the statement WRITE SALARY-RECORD FROM SALARIES (Figure 54, part 4 of 4, line 100). It is recommended that the programmer become familiar with these suggestions and apply them in the writing of 1401 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 AND GO TO FOUND.
   IF ARG = TAB-2 MOVE ENT-2 TO WORK AND GO TO FOUND.
   etc.

   The additional coding effort is offset by dividends

in execution speed for tables with as many as 30 or more entries.

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. The MOVE should be between 01 level records.
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 total of 28 characters 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, 11 additional instruction characters are generated to blank the receiving field.
5. When A and B are unequal length numeric items with identical scaling (same number of decimal places), 14 characters of coding are generated.
6. MOVE A TO B causes 1401 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 MCM 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.
7. 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.
8. 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 1401 edit word.
9. Avoid editing functions which cannot be handled by the 1401 instruction set directly; COBOL zeros, floating plus or minus, DB, and single plus. A special subroutine is called to handle these cases.
10. 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 30 characters.
2. ROUNDED usually generates about 21 additional characters of object code.
3. 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.
4. 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, 1401 COBOL allocates 40 digit accumulators for the storage of temporary results. For example, COMPUTE A ROUNDED = (B * C * D – E) / F, with a varying amount of decimal places, generates about 160 characters of instructions plus 3 X 40 = 120 positions of temporary accumulators. For the equivalent MULTIPLY, SUBTRACT, DIVIDE sequence a total of about 140 positions of storage are used for the instructions and fields.

   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 preceeding 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 1401 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.
7. In order to change the date specified for an input file for label checking purposes, ENTER AUTOCODER and issue a RDLIN IAXX macro. If the file is the *n*th FD in the program, then XX = n+9+m where m is the number of Autocoder names in the SPECIAL-NAMES section.
8. 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.

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 (Figure 54).
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.
10. The Exponentiation-1 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. 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 Exponentiation-2 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.

IBM

COBOL PROGRAM SHEET

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

PAGE 1 3 | PROGRAM SALARIES - IBM-1401 SAMPLE | SYSTEM 1401 | SHEET 01 OF 4
0001 | PROGRAMMER J. JONES | DATE | IDENT. 73 SALARIES 80

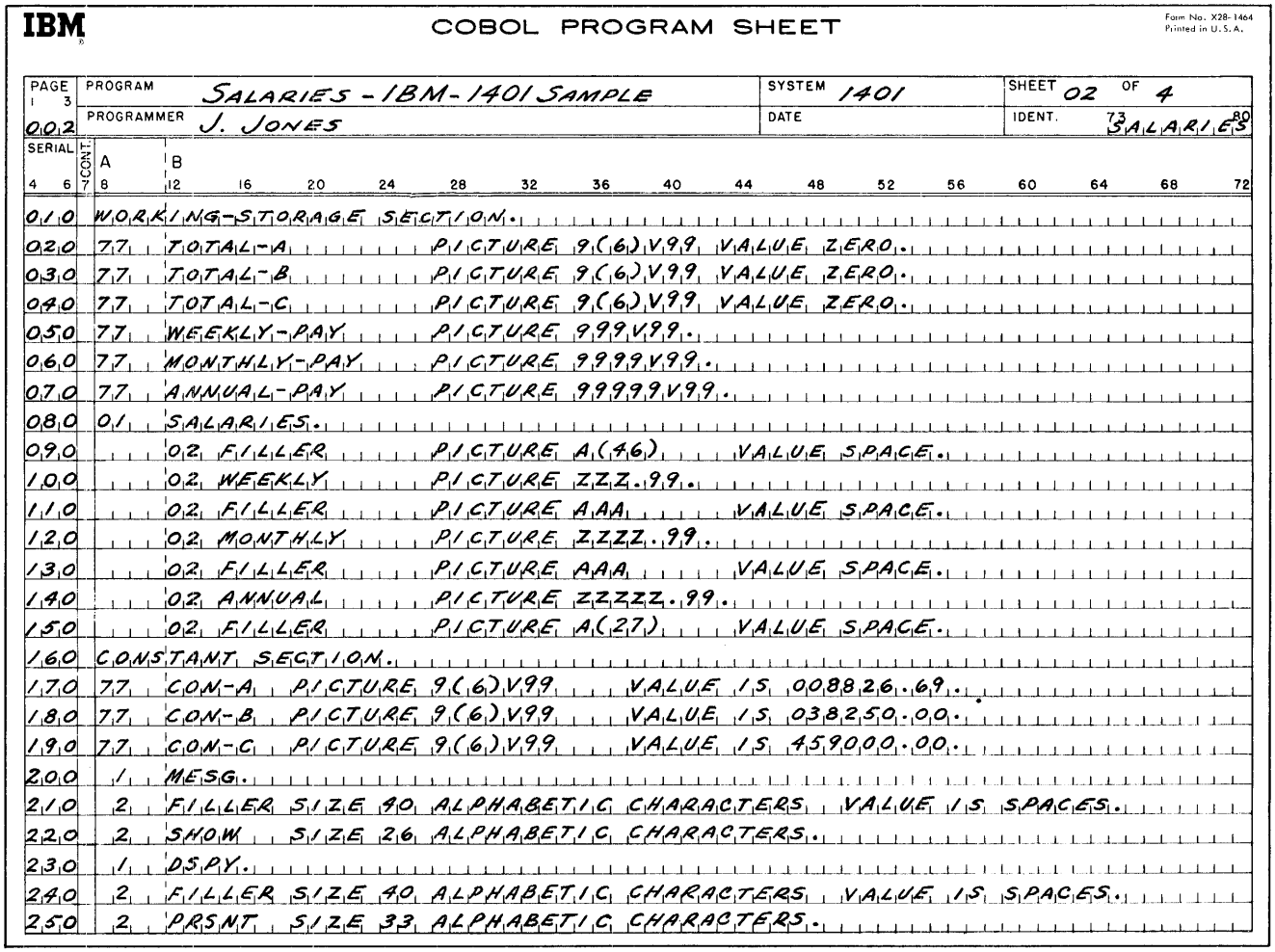
SERIAL 4 6 | CONT. 7 | A 8 | B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72

```
0010 IDENTIFICATION DIVISION.
0020 PROGRAM-ID. 'SALARIES-IBM-1401 SAMPLE'.

0030 ENVIRONMENT DIVISION.
0040 CONFIGURATION SECTION.
0050 SOURCE-COMPUTER. IBM-1401
0060     MEMORY SIZE 8000 CHARACTERS
0070     NO-RELEASE
0080     NO-PRINT-STORAGE.
0090 OBJECT-COMPUTER. IBM-1401
0100     MEMORY SIZE ADDRESS 400 THRU 8000
0110     NO-OVERLAP
0120     NO-PRINT-STORAGE.
0130 INPUT-OUTPUT SECTION.
0140 FILE-CONTROL.
0150     SELECT SALARY-FILE
0160     ASSIGN 1403-P.

0170 DATA DIVISION.
0180 FILE SECTION.
0190 FD  SALARY-FILE
0200     LABEL RECORDS ARE OMITTED
0210     DATA RECORD IS SALARY-RECORD.
0220 01  SALARY-RECORD SIZE IS 100 ALPHANUMERIC DISPLAY CHARACTERS.
```

Figure 54. COBOL Sample, Part 1 of 4



IBM

COBOL PROGRAM SHEET

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

PAGE 1 3 | PROGRAM SALARIES - IBM-1401 SAMPLE | SYSTEM 1401 | SHEET 02 OF 4
002 | PROGRAMMER J. JONES | DATE | IDENT. 73 SALARIES 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
010			WORKING-STORAGE SECTION.
020		77	TOTAL-A PICTURE 9(6)V99 VALUE ZERO.
030		77	TOTAL-B PICTURE 9(6)V99 VALUE ZERO.
040		77	TOTAL-C PICTURE 9(6)V99 VALUE ZERO.
050		77	WEEKLY-PAY PICTURE 999V99.
060		77	MONTHLY-PAY PICTURE 99999V99.
070		77	ANNUAL-PAY PICTURE 999999V99.
080		01	SALARIES.
090			02 FILLER PICTURE A(46) VALUE SPACE.
100			02 WEEKLY PICTURE ZZZ.99.
110			02 FILLER PICTURE AAA VALUE SPACE.
120			02 MONTHLY PICTURE ZZZZ.99.
130			02 FILLER PICTURE AAA VALUE SPACE.
140			02 ANNUAL PICTURE ZZZZZ.99.
150			02 FILLER PICTURE A(27) VALUE SPACE.
160			CONSTANT SECTION.
170		77	CON-A PICTURE 9(6)V99 VALUE IS 008826.69.
180		77	CON-B PICTURE 9(6)V99 VALUE IS 038250.00.
190		77	CON-C PICTURE 9(6)V99 VALUE IS 459000.00.
200		1	MESG.
210		2	FILLER SIZE 40 ALPHABETIC CHARACTERS VALUE IS SPACES.
220		2	SHOW SIZE 26 ALPHABETIC CHARACTERS.
230		1	DSPY.
240		2	FILLER SIZE 40 ALPHABETIC CHARACTERS VALUE IS SPACES.
250		2	PRSNT SIZE 33 ALPHABETIC CHARACTERS.

Figure 54. COBOL Sample, Part 2 of 4

IBM

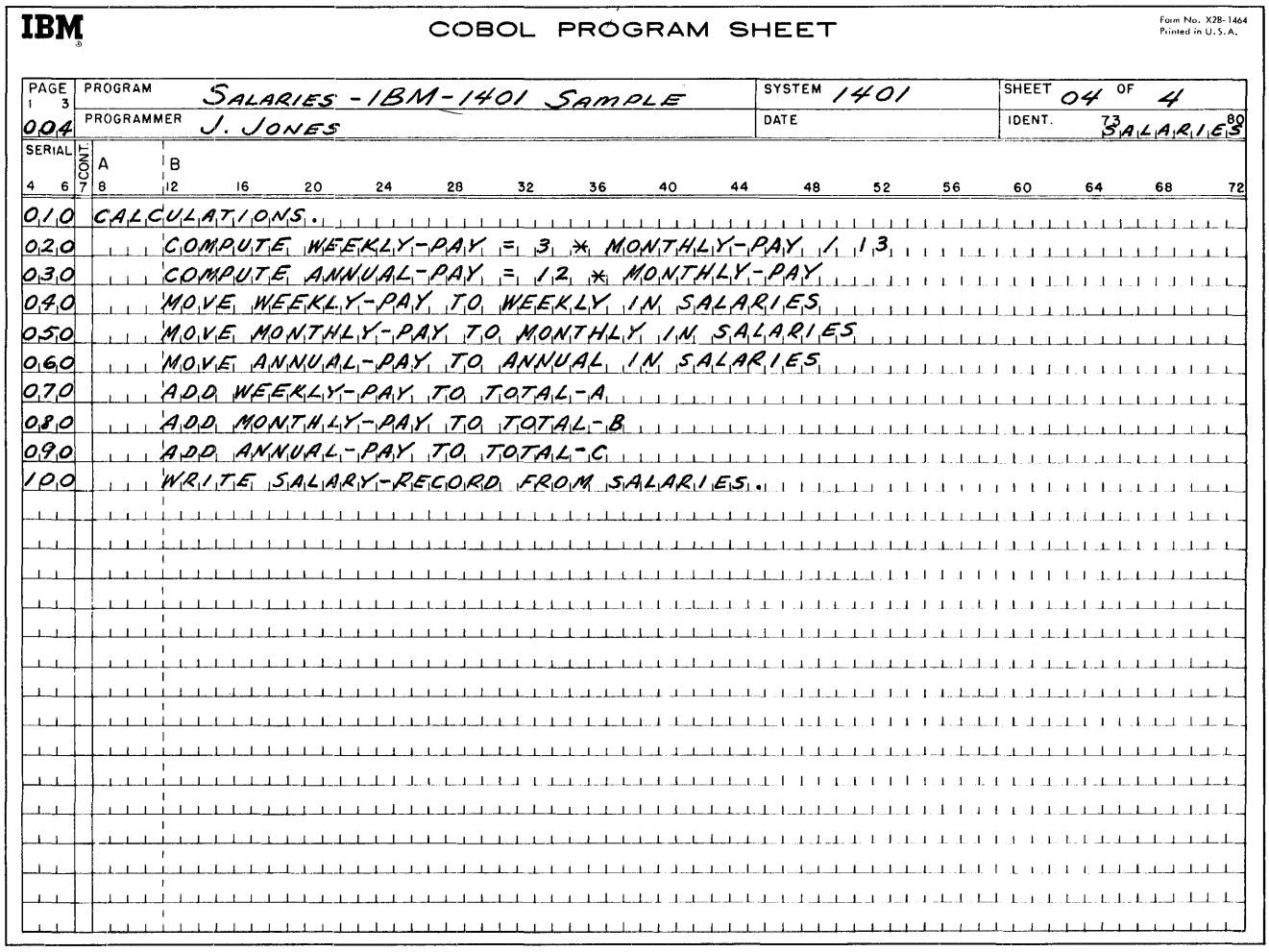
COBOL PROGRAM SHEET

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

PAGE 003 | PROGRAM SALARIES - IBM-1401 SAMPLE | SYSTEM 1401 | SHEET 03 OF 4
PROGRAMMER J. JONES | DATE | IDENT. SALARIES

SERIAL 4 6	CONT 7	A 8	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
010		1	HEADING.
020		2	FILLER SIZE 46 ALPHABETIC CHARACTERS VALUE IS SPACES.
030		2	WEEKLY SIZE 6 ALPHABETIC VALUE IS 'WEEKLY'.
040		2	FILLER SIZE 3 ALPHABETIC VALUE IS SPACES.
050		2	MONTHLY SIZE 7 ALPHABETIC VALUE IS 'MONTHLY'.
060		2	FILLER SIZE 3 ALPHABETIC VALUE IS SPACES.
070		2	ANNUAL SIZE 6 ALPHABETIC VALUE IS 'ANNUAL'.
080		2	FILLER SIZE 29 ALPHABETIC VALUE IS SPACES.
090		PROCEDURE DIVISION.	
100		START. OPEN OUTPUT SALARY-FILE.	
110			WRITE SALARY-RECORD FROM HEADING BEFORE ADVANCING 2 LINES.
120			PERFORM CALCULATIONS
130			VARYING MONTHLY-PAY
140			FROM 500
150			BY 10
160			UNTIL MONTHLY-PAY IS GREATER THAN 10000.
170			IF TOTAL-A = CON-A AND TOTAL-B = CON-B AND TOTAL-C = CON-C
180			MOVE 'TABLE VALUES ARE CORRECT' TO SHOW
190			WRITE SALARY-RECORD FROM MESG AFTER ADVANCING 2 LINES
200			ELSE
210			MOVE 'TABLE VALUES ARE NOT CORRECT' TO PRSNT
220			WRITE SALARY-RECORD FROM DSPY AFTER ADVANCING 2 LINES.
230			CLOSE SALARY-FILE.
240			STOP RUN.

Figure 54. COBOL Sample, Part 3 of 4



IBM

COBOL PROGRAM SHEET

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

PAGE 1 3	PROGRAM SALARIES - IBM-1401 SAMPLE	SYSTEM 1401	SHEET 04 OF 4
004	PROGRAMMER J. JONES	DATE	IDENT. 73 SALARIES 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
010		CALCULATIONS.	
020		COMPUTE WEEKLY-PAY = 3 * MONTHLY-PAY / 13	
030		COMPUTE ANNUAL-PAY = 12 * MONTHLY-PAY	
040		MOVE WEEKLY-PAY TO WEEKLY IN SALARIES	
050		MOVE MONTHLY-PAY TO MONTHLY IN SALARIES	
060		MOVE ANNUAL-PAY TO ANNUAL IN SALARIES	
070		ADD WEEKLY-PAY TO TOTAL-A	
080		ADD MONTHLY-PAY TO TOTAL-B	
090		ADD ANNUAL-PAY TO TOTAL-C	
100		WRITE SALARY-RECORD FROM SALARIES.	

Figure 54. COBOL Sample, Part 4 of 4

IBM

COBOL PROGRAM SHEET

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

PAGE	PROGRAM	SYSTEM	SHEET	OF
001	1401 COBOL SAMPLE	1401	1	4
	PROGRAMMER: J. JONES	DATE	IDENT.	SALARIES

SERIAL 4 6	CONT. 7	A 8	B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
010			IDENTIFICATION DIVISION.
020			PROGRAM-ID. '1401 COBOL SAMPLE'.
030			ENVIRONMENT DIVISION.
040			CONFIGURATION SECTION.
050			SOURCE-COMPUTER.
060			IBM-1401
070			MEMORY SIZE 8000 CHARACTERS.
080			OBJECT-COMPUTER.
090			IBM-1401
100			MEMORY SIZE ADDRESS 400 THRU 8000
110			NO-OVERLAP.
120			INPUT-OUTPUT SECTION.
130			FILE-CONTROL.
140			SELECT SALARY-FILE
150			ASSIGN TO 1403-P.
160			DATA DIVISION.
170			FILE SECTION.
180			FD SALARY-FILE
190			LABEL RECORDS ARE OMITTED
200			DATA RECORDS ARE HEADING-RECORD, SALARY-RECORD, MESSAGE-RECOR
210	-		D.

Figure 55. Second COBOL Sample, Part 1 of 4

IBM

COBOL PROGRAM SHEET

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

PAGE 1 3 | PROGRAM 1401 COBOL SAMPLE | SYSTEM 1401 | SHEET 2 OF 4
0,0,2 | PROGRAMMER J. JONES | DATE | IDENT. 73 SALARIES 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
0,1,0		01	HEADING-RECORD.
0,2,0			02 FILLER PICTURE X(50).
0,3,0			02 WEEKLY-HEADING-LINE PICTURE A(6).
0,4,0			02 FILLER PICTURE X(5).
0,5,0			02 MONTHLY-HEADING-LINE PICTURE A(7).
0,6,0			02 FILLER PICTURE X(6).
0,7,0			02 ANNUAL-HEADING-LINE PICTURE A(6).
0,8,0			02 FILLER PICTURE X(52).
0,9,0		01	SALARY-RECORD.
1,0,0			02 FILLER PICTURE X(50).
1,1,0			02 WEEKLY-DETAIL-LINE PICTURE ZZZ.ZZ.
1,2,0			02 FILLER PICTURE X(5).
1,3,0			02 MONTHLY-DETAIL-LINE PICTURE ZZZZ.ZZ.
1,4,0			02 FILLER PICTURE X(5).
1,5,0			02 ANNUAL-DETAIL-LINE PICTURE ZZZZZ.ZZ.
1,6,0			02 FILLER PICTURE X(51).
1,7,0		01	MESSAGE-RECORD.
1,8,0			02 FILLER PICTURE X(52).
1,9,0			02 MESSAGE PICTURE A(28).
2,0,0			02 FILLER PICTURE X(52).
2,1,0		WORKING-STORAGE SECTION.	
2,2,0		77	HASH-TOTAL-COUNTER-WEEKLY PICTURE 9(6)V99 VALUE IS ZERO.
2,3,0		77	WEEKLY-BUCKET REDEFINES HASH-TOTAL-COUNTER-WEEKLY
2,4,0			PICTURE X(8).

Figure 55. Second COBOL Sample, Part 2 of 4

IBM

COBOL PROGRAM SHEET

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

PAGE 003 | PROGRAM 1401 COBOL SAMPLE | SYSTEM 1401 | SHEET 3 OF 4
PROGRAMMER V. JONES | DATE | IDENT. SALARIES

```
SERIAL  CONT.  A  B
4 6     7      8  12  16  20  24  28  32  36  40  44  48  52  56  60  64  68  72
010     77     WASH-TOTAL-COUNTER-MONTHLY  PICTURE 9(6)V99 VALUE IS ZERO.
020     77     MONTHLY-BUCKET REDEFINES WASH-TOTAL-COUNTER-MONTHLY
030                                        PICTURE X(8).
040     77     WASH-TOTAL-COUNTER-ANNUAL   PICTURE 9(6)V99 VALUE IS ZERO.
050     77     ANNUAL-BUCKET REDEFINES WASH-TOTAL-COUNTER-ANNUAL
060                                        PICTURE X(8).
070     77     WEEKLY-PAY                  PICTURE 999V99 VALUE IS ZERO.
080     77     MONTHLY-PAY                 PICTURE 9(4)V99 VALUE IS ZERO.
090     77     ANNUAL-PAY                  PICTURE 9(5)V99 VALUE IS ZERO.
100     CONSTANT SECTION.
110     77     WASH-TOTAL-OF-WEEKLY-PAY    PICTURE X(8) VALUE '00882669'.
120     77     WASH-TOTAL-OF-MONTHLY-PAY   PICTURE X(8) VALUE '03825000'.
130     77     WASH-TOTAL-OF-ANNUAL-PAY    PICTURE X(8) VALUE '45900000'.

140     PROCEDURE DIVISION.
150     ID-INITIALIZE.
160         OPEN OUTPUT SALARY-FILE.
170         MOVE 'WEEKLY' TO WEEKLY-HEADING-LINE.
180         MOVE 'MONTHLY' TO MONTHLY-HEADING-LINE.
190         MOVE 'ANNUAL' TO ANNUAL-HEADING-LINE.
200         WRITE HEADING-RECORD BEFORE ADVANCING 2 LINES.
210         MOVE SPACES TO HEADING-RECORD.
```

Figure 55. Second COBOL Sample, Part 3 of 4

IBM

COBOL PROGRAM SHEET

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

PAGE 004 | PROGRAM: 1401 COBOL SAMPLE | SYSTEM: 1401 | SHEET: 4 OF 4
PROGRAMMER: V. JONES | DATE | IDENT.: SALARIES

```
SERIAL 4 6   CONT. 7   A 8   B 12 16 20 24 28 32 36 40 44 48 52 56 60 64 68 72
010          START-LOOP.
020              PERFORM CALCULATIONS VARYING MONTHLY-PAY FROM 0500.00 BY
030              0010.00 UNTIL MONTHLY-PAY IS GREATER THAN 1000.00.
040              MOVE SPACES TO SALARY-RECORD.
050          TEST-HASH-TOTALS.
060              IF WEEKLY-BUCKET IS EQUAL TO HASH-TOTAL-OF-WEEKLY-PAY
070              AND MONTHLY-BUCKET IS EQUAL TO HASH-TOTAL-OF-MONTHLY-PAY
080              AND ANNUAL-BUCKET IS EQUAL TO HASH-TOTAL-OF-ANNUAL-PAY
090              MOVE 'TABLE VALUES ARE CORRECT' TO MESSAGE
100              WRITE MESSAGE-RECORD AFTER ADVANCING 2 LINES
110              ELSE
120              MOVE 'TABLE VALUES ARE NOT CORRECT' TO MESSAGE
130              WRITE MESSAGE-RECORD AFTER ADVANCING 2 LINES.
140              CLOSE SALARY-FILE. STOP RUN.
150          CALCULATIONS.
160              COMPUTE WEEKLY-PAY = 3 * MONTHLY-PAY / 13.
170              COMPUTE ANNUAL-PAY = 12 * MONTHLY-PAY.
180              MOVE WEEKLY-PAY TO WEEKLY-DETAIL LINE.
190              MOVE MONTHLY-PAY TO MONTHLY-DETAIL-LINE.
200              MOVE ANNUAL-PAY TO ANNUAL-DETAIL-LINE.
210              ADD WEEKLY-PAY TO HASH-TOTAL-COUNTER-WEEKLY.
220              ADD MONTHLY-PAY TO HASH-TOTAL-COUNTER-MONTHLY.
230              ADD ANNUAL-PAY TO HASH-TOTAL-COUNTER-ANNUAL.
240              WRITE SALARY-RECORD.
```

Figure 55. Second COBOL Sample, Part 4 of 4

Index

COBOL (on Tape) Specifications: IBM 1401 (C24-1492-2)

- Is the material:

	Yes	*Satisfactory*	*No*
Easy to read?	☐	☐	☐
Well organized?	☐	☐	☐
Fully covered?	☐	☐	☐
Clearly explained?	☐	☐	☐
Well illustrated?	☐	☐	☐

- How did you use this publication?
  - As an introduction to the subject ☐
  - For additional knowledge of the subject ☐

- Which of the following terms best describes your job?

Customer Personnel		*IBM Personnel*	
Manager	☐	Customer Engineer	☐
Systems Analyst	☐	Instructor	☐
Operator	☐	Sales Representative	☐
Programmer	☐	Systems Engineer	☐
Trainee	☐	Trainee	☐
Other _____		Other _____	

- Check specific comment (if any) and explain in the space below:
  (*Give page number*)
  - ☐ Suggested Change (Page) ☐ Suggested Addition (Page)
  - ☐ Error (Page) ☐ Suggested Deletion (Page)

Explanation:

Space is available on the other side of this page for additional comments.
Thank you for your cooperation.

C24-1492-2

Staple

Fold

Fold

FIRST CLASS
PERMIT NO. 387
ROCHESTER, MINN.

BUSINESS REPLY MAIL
NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

POSTAGE WILL BE PAID BY . . .

IBM Corporation
Systems Development Division
Development Laboratory
Rochester, Minnesota 55901

Attention: Product Publication, Dept. 245

Fold

Fold

Cut Along Line

IBM 1401 Printed in U.S.A. C24-1492-2

IBM®

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

Additional Comments:

IBM 1401 Printed in U.S.A. C24-1492-2

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