FACT

FULLY AUTOMATIC COMPILING TECHNIQUE

A NEW BUSINESS LANGUAGE

Electronic Data Processing

FACT

FULLY AUTOMATIC COMPILING TECHNIQUE

A NEW BUSINESS LANGUAGE

Honeywell

H HONEYWELL *Electronic Data Processing*

The specifications of FACT have been created by Computer Sciences Corporation, 3402 West Century Boulevard, Inglewood, California, under contract to the DATAmatic Division of Minneapolis-Honeywell. Computer Sciences Corporation also has the contract to implement FACT for the Honeywell 800 Data Processing System.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

PREFACE

FACT (Fully Automatic Compiling Technique) is a complete programming system which constitutes a major breakthrough in programming cost reduction. It can be used on any data processor for which it is implemented. In the past, problem preparation for a major data processing application has been exceedingly expensive and time consuming. In some cases, its cost has approximated that of the equipment itself. Initial efforts by many people to reduce this cost led gradually to an assortment of programming aids, each of which contributed a little to making the programmer's task easier. Though the combined effect of these aids has been significant, their very abundance has limited their usefulness. Furthermore, the variation from installation to installation has been unfortunate. Now for the first time, a tool is presented which provides for easy and uniform handling of all aspects of data processing, including input editing, sorting, processing of variable-length records, and report writing.

FACT provides the following major benefits:

1. Drastic reduction of the cost of problem preparation;
2. Significant reduction of the staff required;
3. Significant reduction of time needed to put electronic data processing into operation;
4. Significantly easier program modification to reflect changing requirements;
5. Increased computer use on broader range of jobs; and
6. Interchangeability of programs among different installations.

There are many ways in which FACT accomplishes these effects. Some of these ways are discussed in the following paragraphs.

The programmer provides the compiler with a set of simple inputs which completely describe his problem. FACT creates the program which does the job. All of the machine-oriented problems are solved by the compiler. This is the reason that the programmer's task is so drastically reduced. For example, the following features of FACT greatly reduce tasks which formerly accounted for large amounts of programmer effort.

FIELDS: The natural nouns of data processing are fields; the natural nouns of machines are words. Since these are usually incommensurate, much of the program-

mer's task has been devoted to writing orders which fit fields into words.

DIRECTORIES: The description of a file used to be buried in every program related to that file in the form of shifts, transfers, extractions, substitutions, and address selections. This was a never-ending source of programmer labor. Now the programmer describes his file once; FACT creates a directory which it then uses over and over to create new programs related to that file without the programmer giving it any further thought. The programmer's file description is now concerned only with the information in the file - not how its fields are placed in words. This eliminates all that reprogramming which was formerly required when a file structure was changed.

INPUT-OUTPUT HOUSEKEEPING: A significant portion of any data processing job consists of programs to write and check tape labels, read tapes, check the validity of information read, reread or reconstruct if necessary, and move the records into working position or into position to be written. All of these programs are created by FACT without any attention by the programmer.

RESTART PROCEDURES: Formerly programmers were required to devote a considerable amount of attention to writing their programs in such a way that only a small amount of time would be lost if a program were interrupted for any reason. FACT provides a way to do this with little programmer effort.

GENERATORS: Not the smallest contributions to reduced programming effort are the built-in generators; input editing, sorting and report writing. These make the necessary preparation so simple that it will frequently be possible to conceive a small data processing job in the morning, execute all the necessary preparations, compile a program and make the run the same day. The job might contain input editing, file creation, sorting, merging, record selection and elementary processing, and reporting. Using the best tools formerly available - library routines, linkages, and pieces of manual coding - the programmer would have been very fortunate and skilled to accomplish the same job in a month or two.

SECTION I
INTRODUCTION

FACT translates problem-oriented (or source) language into computer language, creating a program in operating form. The operating program produced by FACT, called the object program, may be placed on punched cards or magnetic tape. The conciseness of FACT language results in a high ratio of machine instructions to source statements. In addition, FACT produces complete printed information about its own operation, including program listings, memory assignments, and diagnostic data pertaining to programming errors encountered during compilation. All of these outputs and aids are expressed in language easily understood by the programmer.

The FACT lexicon is made up of the familiar words of everyday business usage, such as FILE, ENTRY, PROCEDURE, REPORT, DELETE, and UPDATE. Source programs are written by combining lexicon words with the names of data units (files, entries, fields) to form ordinary English sentences and paragraphs. The result is a smooth-flowing language. FACT is offered as a contribution to the development of a common data processing language.

Equipment Requirements

As implemented on the Honeywell 800, FACT can compile with only four magnetic tape units and minimum memory. It can, however, take advantage of more equipment. The programmer uses environment statements to describe the equipment array available for compilation, as well as the array on which the object program is to be run. Each object program is compiled to operate as efficiently as possible with the allotted machine units. A Honeywell 800 System having the minimum configuration of equipment may be used to compile programs for execution on any other Honeywell 800.

FACT Inputs

The inputs required for compilation may include any or all of the following:

(1) Descriptions of the formats of all input data cards to be processed by the object program;

(2) Descriptions of all data files to be created by the object program;

(3) Descriptions of all reports to be prepared by the object program;

(4) Source statements in narrative business English which describe the processing operations to be performed by the object program;

(5) Environment statements which specify the kinds and amounts of equipment (actual machine units) available for both the compilation and execution of the object program;

(6) Control statements which specify the forms of output required, error procedures, and actions to be taken after compilation or during execution of the object program.

The inputs required to compile a specific program depend upon the nature of the program. FACT may be used to prepare many different types of programs at many different levels of complexity. For example, a given object program may perform any or all of the following functions:

(1) Input card reading and editing;

(2) Creation of data files;

(3) Data sorting;

(4) Arithmetic computations;

(5) Updating of the data files; and

(6) Generation of printed or punched reports based on input data, file data or program results.

The specific functions to be performed by a given program and the equipment array allotted for their performance govern the program solution produced by FACT. In particular, the programmer may use environment statements to take advantage of the multi-programming feature of the Honeywell 800. For example, an input editing program is particularly suited for processing in parallel with a reporting program. If sufficient equipment is provided, both may be executed in parallel with a file updating program. Thus, large, complex programs involving input editing, data processing, and reporting may be designed for a three-phase type of operation in which an input file is processed in parallel with the preparation of reports generated by the previous run and the editing of input data for the succeeding run.

FACT Outputs

The primary output from FACT is an operating program. After compilation, this program is on magnetic tape or punched cards. In either case, the programmer may direct that the object program be immediately loaded into the computer and executed. The object program is accompanied by the labels and the directories of all data files which it creates or references.

In addition to the object program, FACT produces the following outputs:

(1) A listing of the narrative source statements;

(2) A detailed map, showing the use of high-speed memory by the object program;

(3) Diagnostic comments describing the number, kinds, and locations of all errors encountered in the source program; (FACT can detect such errors as incorrect spelling, improper use of data names, and illegal file structures.)

(4) Operator instructions requesting specific actions during and immediately after compilation.

(5) New file directories.

The following sections of the manual present detailed descriptions of the various FACT inputs and the manner in which source programs are presented for compilation. The reader may find it helpful to refer to the sample application contained in Section VI for examples of these inputs.

SECTION II
FILES

The basic function of a data processing operation is the creation and manipulation of files. A file is an arrangement of data according to a specific format and usually in a specific sequence. The basic units of data, (e.g., part number, city, name, quantity) are called fields. A number of fields may be so related that for purposes of manipulation it is desirable to give them a collective name. Such a set of related fields is called a group. In the same way, it may be desirable to assign a name to a set of related groups, forming a group of higher rank or level. This process may be extended to form a hierarchy of groups of several ranks. FACT allows the programmer to reference and manipulate individual groups of any rank.

The programmer uses a File Outline Form to assign the names of groups, specify their order within the file and describe their relationships (inclusiveness). The relationships of fields within groups and of groups within higher-ranking groups is specified on the File Outline Form by means of indentation.

Within a group of any rank, the names of all groups of the next lower rank are indented on the file outline by the same amount, which may be one or more columns. The name appearing farthest to the left is the file name and includes all groups and fields in the file. The first group name below and to the right of the file name is called an entry name. Any other group whose name is indented the same amount is also an entry. An entry may include fields and groups. For example, a payroll file might include such fields as DIV-NO, DEPT-NO, and CLOCK-NO. Figure 1 is an outline of such a file which is arranged in wage-type, employee-number sequence. For purposes of clarity, all non-field names are identified in a separate column.

The reader will note in Figure 1 that fields are not always of the lowest rank. On the contrary, fields may occur at any rank up to that of an entry. In this manual, a field is regarded as a special type of group which contains no lower-level information.

Another basic distinction among groups is illustrated in Figure 1. Within a group, certain subgroups (e.g., NUMBER) occur a fixed number of times. Such groups are called primary. Other subgroups (e.g., EMPLOYEE-RECORD, HOURS, BASESHIFT, etc.) occur an indefinite number of times. Such groups are called

```
PAYROLL                                          (file name)
    *HOURLY                                      (entry name)
      *EMPLOYEE-RECORD                           (group name)
        NUMBER                                   (group name)
            DIV-NO
            DEPT-NO
            CLOCK-NO
        BASE WAGE
        OVERTIME-PERCENTAGE-BONUS
       *HOURS                                    (group name)
          TOTAL-HOURS
         *BASESHIFT                              (group name)
              PER-DAY
         *NIGHTSHIFT                             (group name)
              PER-DAY
    *SALARIED                                    (entry name)
      *EMPLOYEE-RECORD                           (group name)
        NUMBER                                   (group name)
            DIV-NO
            DEPT-NO
            EMPLOYEE-NO
        SALARY
        OVERTIME-RATE
        HOURS-WORKED
```

Figure 1.

secondary and are so designated on the file outline by placing an asterisk in front of their names. The asterisk is not considered part of the name for indentation purposes. Note that any group which contains one or more secondary subgroups must itself be secondary.

The distinction between primary and secondary groups is important since it determines the manner in which information is stored on magnetic tape and in memory. Primary groups are stored in memory throughout the processing of the related higher-ranked group. Secondary groups are stored in memory a fixed number at a time. This subject is described in greater detail under "Data Placement" (page 22) and "Use of Tape Files" (page 27).

File Description

Object programs produced by FACT may create original files from input data. Such programs may also process existing files and create new files as a result of such processing. In particular, an object program may create a working file which exists only in memory and is not recorded on tape. Such a file is called an internal file. Tape files may be either packed or unpacked. A packed file occupies less space on tape than an unpacked file; however, it requires more complicated input-output routines to handle it. Both packed and unpacked files may be specified as reversible, which means that they may be read (but not written) in both the forward and the reverse direction.

One of the basic tasks involved in the preparation of a FACT source program is the description of the structures of new internal and tape files to be used by the object program. The structure of each such file is presented on a special form called the File Outline Form, which is shown in Figure 2. One or more lines (or rows) on this form are used to describe each group from the file itself down to the lowest-level field. The hierarchical relationships between the various file elements are represented by relative amounts of indentation, just as they were in Figure 1.

Each line of the File Outline Form becomes the contents of a special punched card called a File Outline Descriptor. A deck of File Outline Descriptors, then, contains the complete description of a file structure. (To prevent any confusion between punched cards which provide input information to the Compiler and those which provide input data to the object program, the former will be consistently referred to as "descriptors" and the latter as "cards". FACT input includes several other types of descriptors, which are described in later sections.)

FACT uses the information obtained from the deck of File Outline Descriptors to generate a directory of the file being created. The file directory is stored with the file which it describes and also in a library of file directories on the program tape. It is not necessary to prepare a File Outline for an existing file which is to be processed by an object program since FACT can reference the library of directories to obtain the necessary structural information about the desired file.

File Outline Form

As noted above, each information unit in a file is described by one or more lines on the File Outline Form. The descriptive information is written in fixed fields, as shown in Figure 2. If the information corresponding to any fixed field exceeds the capacity of that field, additional lines (called continuation lines) are used to complete

Honeywell

Electronic Data Processing

FILE OUTLINE FORM

Title ..... PAYROLL FILE ..... Revision .....

Prepared By ..... For Program .....

Date ..... Checked By .....

Remarks .....

FACT

	0	SERIAL NUMBER	NORMAL ALLOTTED LENGTH	MAX. LENGTH OR NUMBER OF APPEARANCES	MODE	DECIMAL POINT	ROUND	FILE ENTRY	NAME (DECREASING RANK →)
1	0	1			RU				PAYROLL
2	0	2							*HOURLY
3	0	3							*EMPLOYEE-RECORD
4	0	4							NUMBER
5	0	5		1	D				DIV-NO
6	0	6		2	D				DEPT-NO
7	0	7		4	D				CLOCK-NO
8	0	8		3	D	02			BASEWAGE
9	0	9		4	D	02			OVERTIME-PERCENTAGE-BONUS
10	0	10							*HOURS
11	0	11		3	D	01			TOTAL-HOURS
12	0	12							*BASESHIFT
13	0	13		3	D	01			PER-DAY
14	0	14							*NIGHTSHIFT
15	0	15		3	D	01			PERDAY
16	0	16							*SALARIED
17	0	17							*EMPLOYEE-RECORD
18	0	18							NUMBER
19	0	19		1	D				DIV-NO
20	0	20		2	D				DEPT-NO

FORM NO. T1204

Figure 2.

the description of that information unit. The information contained in a continuation line is punched in a continuation descriptor. The use of the File Outline Form is clarified in Figure 2 by repeating a portion of the simple file described earlier.

DESCRIPTOR TYPE -- column 1: Every descriptor is identified as to type by the contents of this column. A File Outline Descriptor is identified by the presence of the letter O or the numeral zero punched in column 1.

SERIAL NUMBER -- columns 2-6: Each line of the File Outline Form may be assigned a serial number by the programmer. If these numbers are assigned, the Compiler may be directed to check their sequence, listing any sequence errors detected or else sorting the descriptor information on these numbers. Unserialized descriptors must be presented to the Compiler in the correct sequence. If the description of any information unit in the file requires the use of one or more continuation lines, each continuation line is designated by the presence of an X overpunch in column 6. If serial numbers are assigned, the first or base line and all continuation lines relating to a specific information unit are assigned numbers in sequence. (If decimal numbers up to 99999 do not suffice, B C D E F or G may be used in any digit position, according to the sequence 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, B, C, D, E, F, G.)

NORMAL ALLOTTED LENGTH -- columns 7-9: In order to minimize space on tape for internally created variable-length fields, the programmer specifies here a length which is sufficient to accommodate most anticipated values. For a fixed-length field, these columns are left blank.

MAXIMUM LENGTH OR NUMBER OF APPEARANCES -- columns 10-12: The length of an internally created field is stated in these columns. If the length of such a field is variable, the maximum length is stated here. The lengths of other fields are available to FACT from the description of the input data and need not be repeated on the File Outline Form.

In the case of primary groups, columns 10-12 are used to state the number of appearances of such groups within the related higher-level group. In the case of secondary groups, they may be used to specify the number of such groups which are desired to be simultaneously available in memory. If the latter is not specified, secondary groups are made available one at a time.

If columns 7-9 and/or 10-12 are used, their contents must be written justified to the right. If they are not used, FACT assumes that the information is otherwise available to it.

MODE -- columns 13-14: The data to be processed by an object program may be in any of a number of different modes, as specified by the following codes:

Code	Mode	Legal Characters
A	Alphabetic	A-Z
AN	Alphanumeric	A-Z and 0-9
AS	Alphabetic, Numeric and Sign	A-Z, 0-9, 11, and 12 (or - and +)
H	Hollerith	Any legitimate punch
NH	Numeric Hollerith	0-9
SP	Single Punch	0-9, 11, and 12
D	Decimal[1]	0-9 and possible zone punches for signs
CD	Decimal with Check Digit[2]	0-9
HD	Hexadecimal[3]	0-9 and B-G
OC	Octal[3]	0-7

The mode of any field which is filed from input data cards is described as part of the input data description and need not be described on the File Outline Form. Columns 13-14 need only be completed for fields which are created internally. Single-character codes may be written in either column 10 or 11; i.e., (A, blank) is equivalent to (blank, A).

In the case of the file name, the mode columns are used to state whether the file is to be Packed (P), Unpacked (U), Reversible Packed (RP), Reversible Unpacked (RU), or Internal (I).

DECIMAL POINT -- columns 15-16: In decimal fields which are internally created by the object program, the position of the decimal point may be indicated by a number in columns 15-16. This is called scaling. In this manner, a numeric quantity may be represented internally as an integer and treated by the object program as a number with a decimal point. If the contents of a field are to be internally scaled, the position of the decimal point is indicated by a number written in these columns and justified to the right. If these columns are blank and no scaling is specified elsewhere, the decimal point is assumed to be to the right of the low-order digit. If

1. A decimal field in the Honeywell 800 may contain up to eleven digits.
2. A check decimal field or an unsigned decimal field in the Honeywell 800 may contain up to twelve digits, including the check digit.
3. A hexadecimal or octal field may be up to one machine word in length.

scaling is specified, the direction of scaling is indicated by an overpunch in column 16. A 12 overpunch indicates positive scaling or a decimal point to the right of the above assumed position. An 11 overpunch indicates negative scaling or a decimal point to the left of the above assumed position. If scaling is specified, but column 16 does not contain an overpunch, negative scaling will occur. For example, the value stored in a decimal field is 47891. If columns 15-16 contain the punches 03 or $0\overline{3}$ (where $\overline{3}$ represents a 3 punch with an 11 overpunch), the scaled value of this field is 47.891. If columns 15-16 contain the punches $0\overset{+}{2}$ (where $\overset{+}{2}$ represents a 2 punch with a 12 overpunch), the scaled value of this field is 4,789,100. As in the case of length and mode, the scale of fields read from input cards is specified as part of the card input description.

ROUND -- column 17: Sometimes it is desirable to retain in a file only a portion of the significant digits of a decimal or check decimal field. A field read from punched cards or from an input tape file may contain seven significant digits, whereas five of these digits are sufficient for the purposes of a particular output file being created. If column 17 of the File Outline Form contains an R punch or is blank, the low-order digit of the shortened field will be rounded according to the value of the digits dropped. If this column contains a T punch, the field will be truncated, i.e., the resulting low-order digit will be unmodified regardless of the value of the digits dropped.

NAME -- columns 18-80: The name of the file, entry, group, or field being described is punched in these columns. The position of the left-most character in the name denotes the rank of the corresponding unit. The highest-ranking group (which is the file) is always described in the first line and its name is punched starting in column 18. A name which starts in column 19 is either an entry or a field at the entry level.

If two or more files are to have identical file structures, the names of all such files may be written on consecutive lines at the top of the file outline which describes their structure. If one or more new files are to have the same structure as a file which has been previously described, the name of the earlier file may be written at the top of the file outline, followed on succeeding lines by the names of all new files which are to have the same structure. The final name is followed by a blank line (i.e., a blank File Descriptor) which signifies that the desired file structure is already available and need not be repeated.

In some cases, a processing run simultaneously involves two or more instances of the same file. For example, an updating run may read an existing file and create

an updated file having the same name (often to be used as input to the next updating run, etc.). The name of such a file (or files) may be modified on the file outline by writing adjectives after the file name. The programmer writes a different adjective to represent each of the files of that name to be manipulated simultaneously. The file name and all modifying adjectives are separated by commas. For example, the name columns of a file outline may contain:

MASTERFILE, ANNUAL, MONTHLY, WEEKLY

where MASTERFILE is the name assigned to three different files. Any of these three files may be specifically referenced in the source program by using the modified name (i.e., WEEKLY MASTERFILE).

The name of an information unit is assigned by the programmer and is used by the Compiler to cross-reference that unit in the input data, the various files involved, the source statements, and the required reports. The programmer must take care to identify a given unit by the same name wherever this unit is referenced, with the exception that a name may be followed by one or more abbreviations in parentheses and then referenced by either the full name or any abbreviation so listed. A name may be composed of any number of letters (A-Z), numerals (0-9), and hyphens, except that the first character of a name must be a letter. Imbedded blanks are not allowed; consequently, multi-word names must be connected by hyphens, as in the following:

INVENTORY (INVENT)
OLD-MASTER-FILE (O), (OMF)
CITY-AND-STATE
POLICY-NUMBER (POLNUM), (NO), (NU).

As indicated above, a name may be followed by any number of abbreviations. These are individually enclosed in parentheses and may be separated by commas if desired. The length of a name is unrestricted. If the complete name (including all abbreviations) is too long to write on a single line, any required number of continuation lines may be used to complete the information.

The name of a field may also be followed by one or more numeric or non-numeric literals. A literal is an explicit expression of the value of a field as opposed to a value which is implicitly expressed by referencing the field name. In other words, instead of referencing a field called PRICE by name, the programmer might reference an explicit value of this field, such as 124.95. Such a value is called a numeric literal. In the same manner, a field called TITLE might represent the title of a

report. The programmer might reference a specific report title, such as "Summary of Operating Costs", which would be a non-numeric literal. Numeric literals may be entered directly in the name columns of the file outline. Since a name is not allowed to start with a numeral, there is no danger of confusion. However, non-numeric literals must be distinguished from names by writing them preceded and followed by two dots (..). Literals may be used on a file outline for any of several purposes. If the value of a field is constant throughout a program, the value may be written as a literal following the name of the field, as follows:

```
PI              3.1416
COMPTROLLER     ..DAVID S. PARKER ..
```

The programmer's source statements may then reference the name of such a field in order to utilize the stated literal value of the field.

Another use of literals is the relating of codes which appear in input data to adjectives which can then be used to represent these codes in the source program, as in the following example:

```
WAGE-SCHEDULE ..H..(HOURLY) ..W..(WEEKLY) ..M..(MONTHLY)
```

In this example, the three literals H, W, and M are the only defined values for the field WAGE-SCHEDULE. These values may be referenced in the source program by means of the related adjectives. This type of literal expression facilitates the writing of conditional source statements. The presentation and use of literals is described in greater detail in Section III.

SECTION III
SOURCE LANGUAGE

The object program produced by FACT is an efficient machine-language translation of the source program statements which the programmer writes. This object program is based, in addition, on programmer statements of the directories of the data files to be processed, the equipment configuration available for both compilation and execution, the formats of any reports to be printed, and the desired error procedures and operator actions during compilation. The program statements, themselves, constitute a straight-forward description in everyday business English of the operations to be performed, presented in familiar paragraph format. The ingredients of these statements are the basic verbs, nouns, and connectives of the Compiler lexicon, the names of information units to be processed, ordinal and cardinal numbers, and explicit quantities and names called literals. The following types of source statements may be included in a program:

Imperative
Control
Conditional
Definition

The use of these types of statements is illustrated in the sample application described in Section VI.

In addition to the above, the source program language contains several unusual and powerful features. These include:

(1) A generalized updating routine which can be tailored to a specific set of updating functions merely by stating the master file, detail file, and key names, and the procedures to be followed at a set of standard junctures;

(2) A generalized sort routine which requires only the file name and, in some cases, the names of the sort keys, in major-to-minor sequence;

(3) A report generator which requires only the format of the desired report and its included lines and fields; and

(4) The ability to handle information in tabular form and to obtain any value from such a table when called for by the program.

Any of these routines and special features can be used by means of a single source language statement.

Throughout this section, numerous examples of source language are presented for illustrative purposes. These examples adhere to several editorial conventions which are not a part of the source language and are not used during actual program preparation.

(1) Lexicon words are not underlined; all other words, which are underlined, may be freely changed to conform to a particular application;

(2) Lexicon words which may be omitted without change of meaning are enclosed in parentheses;

(3) Typical names or symbols (e.g., A, B, C) are used in some examples. Elsewhere, the possible presence of a name is indicated by an underlined blank. Such names, symbols, and blanks in no way connote any definition of or restriction on the names which the programmer may use.

Names

The name of any information unit may contain any number of alphabetic characters and decimal digits, except that it must start with an alphabetic character. Imbedded blanks are not permitted; hyphens must be used to connect multi-word names. The programmer may also assign a name to any paragraph or procedure. Names of this type need not start with a letter; in other words, only paragraphs and procedures may be numbered.

Modification of Names

Names of groups at any level may be modified by either adjectives or prepositional phrases.

ADJECTIVES: A name may be preceded by one or more adjectives, written in order from the least limiting to the most limiting. In other words, the adjective which defines the broadest category is written first; that which defines the narrowest category immediately precedes the name. For example, a file called WAREHOUSE may contain an entry called PART which contains a field called NUMBER. This field, then, is completely described by the modified name:

WAREHOUSE PART NUMBER.

PREPOSITIONAL PHRASES: A second method of modifying a name is by the use of prepositional phrases involving the prepositions IN or OF. Modifiers of this type

must be written in order from the narrowest to the broadest category. Thus the above example, if written with phrasal modifiers, would appear as:

NUMBER OF PART IN WAREHOUSE.

(The reader is reminded that the non-underlined words are to be found in the Compiler lexicon.)

Any combination of adjectival and phrasal modification is acceptable, provided only that each type of modifier is presented in the correct sequence. Thus the following are both correct methods of specifying the above field:

PART NUMBER IN WAREHOUSE;
NUMBER OF WAREHOUSE PART

Abbreviations

Any name may be followed in the source program by one or more abbreviations, each individually enclosed in parentheses. An abbreviation may not include embedded blanks. An abbreviation which follows a modified name applies to the name as modified and not merely to the name itself.

An information unit may be referenced by its name or by any properly assigned abbreviation. Alternate abbreviations may be assigned at the same point or at different points in the source program. An abbreviation assigned to a group name must not be reassigned within the same program. An abbreviation assigned to a field name, on the other hand, may be reassigned to a different field name at another point in the source program. When this type of dynamic abbreviating is used, an abbreviation retains its assigned meaning during object program execution up to the time that a new assignment is reached.

Literals

A literal is an explicit use of the actual value of an information unit, as opposed to the use of the name of that unit. Literals may be either numeric or non-numeric. The former may be written in the source statements in their natural form.

Non-numeric literals must be distinguished from other source language words, since they are handled explicitly as written and not used to refer to other information. This may be accomplished by preceding and following a non-numeric literal with quote symbols ("). Since this symbol is not available on most card punches, the symbol .. preceding and following the literal is interpreted in the same manner. For example, the statement:

REPLACE TITLE BY ..SUMMARY OF OPERATING COSTS...

directs the Compiler to insert the words "Summary of Operating Costs" in the field named "Title". Notice that the literal at the end of a sentence is followed by two dots which comprise the literal marker and a third dot which denotes the end of the sentence. If this is liable to result in any confusion, the entire literal with its literal marker may be enclosed in parentheses without altering the meaning of the statement. A non-numeric literal may never include the symbol which is used as the literal marker. In other words, no literal containing two successive dots may be written if this symbol is used, no literal containing a quote symbol may be written if quote symbols are used, and no literal containing two successive dots followed by a closing parenthesis may be written if dots and parentheses are used. A literal may not end with a period if two periods are used as a literal marker.

Numbers

Cardinal numbers, in addition to being recognized as numeric literals, are recognized from zero to billion when written out according to the spellings as given in the Table of Numbers, Webster's New Collegiate Dictionary, except for the word "Naught". Ordinal numbers from first to billionth may be used as adjectives when abbreviated according to the above table, except for the short forms 2d and 3d, which are not permitted. A generalized or variable ordinal is written by enclosing the name of the quantity in parentheses and appending the letters TH after the closing parenthesis, e.g.:

(I)TH, or (M PLUS N)TH.

Numeric Quantities

Numeric quantities may be specified as field names, numeric literals, or arithmetic expressions which are combinations of either or both. Such combinations may be formed by the use of the lexicon words or symbols:

PLUS or +
LESS or MINUS or -
TIMES or MULTIPLIED BY or *
OVER or DIVIDED BY or /.

Parentheses may be used to denote the distribution of these operations. If no parentheses are included, all multiplications and divisions are performed before additions and subtractions. In other words, in the absence of parentheses the words TIMES and OVER (and their substitutes) are not distributive. For example, the quantity

A TIMES B PLUS C or A * B + C

is equivalent to C plus the product of A and B. Likewise, the quantity

A / B - C * D

is equivalent to the difference between the quotient of A over B and the product of C and D. Any other distribution of operations is expressed by the inclusion of parentheses, as follows:

A TIMES (B PLUS C)

A / ((B - C) * D)

These quantities are equivalent, respectively, to A times the sum of B and C, and A divided by the product of B minus C and D.

Characters

The programmer may refer to an individual character, digit, or letter of a field by an expression of the form:

FIRST DIGIT OF POLICY NUMBER

or (24)TH LETTER OF NAME

or THIRD CHARACTER OF ADDRESS,

counting from the left.

Sentences

Program statements take the familiar form of sentences, complete with terminal periods and commas where customary. Construction is normally idiomatic, with the exception of some plural forms and some definitions of combined conditions described below. Words in a series may be separated by commas or by the word "AND". If the latter is used, commas may also be written to improve readability, but they are not required. Successive words must be separated by at least one space. It is suggested that this rule be followed even where punctuation marks separate words. An opening parenthesis must be preceded by at least one space except where it is the first character of a line. The Compiler does not distinguish between one space and several spaces, except in the determination of indentations.

No line in the source language may end in the middle of a word without hyphenating. If a hyphenated multi-word name is written with its hyphen at the end of a line, a second hyphen is required at the beginning of the next line. No line may begin with a punctuation mark other than a hyphen, opening parenthesis, or a literal marker. Every sentence, and hence every paragraph, must be followed by a period. Two or more simple sentences may be combined using the connective AND to form a compound sentence.

Source Program Statement Form

Source statements are written by the programmer using the Source Program Statement Form, shown in Figure 3, page 19. The contents of this form are used to punch Source Program Descriptors, identified by a P punched in column 1. Columns 2-6 may be used to specify serial numbers, exactly as on File Outline Descriptors (see page 8). The balance of the form (columns 7-80) contains the statements themselves, according to the paragraph technique described below.

Paragraphs

Source sentences should be combined, as nearly as possible, into logically integrated paragraphs, as in normal English. The efficiency of segmentation in the object program is related to the effectiveness of paragraphing in the source program.

PARAGRAPH NAMES: The programmer may assign a name to a paragraph if he desires to refer elsewhere to that paragraph as a whole. A paragraph name takes the same form as any other name except that it may begin with a number. Alternate names or abbreviations may be defined using parentheses, e.g.:

PAY-COMPUTATION (P-C), (PAY).

The comma is optional.

The first line of a paragraph (beginning with the name, if any) is indented by at least one space with respect to any succeeding lines of the paragraph. The first lines of all paragraphs within a given program must be indented by the same amount. Alternatively, the paragraph name may be made to stand out by writing it on a separate card preceding the paragraph. In this case, the position of the paragraph name on the card is irrelevant.

These two ways of presenting paragraph names are illustrated by the following two examples.

```
    TAX. IF 13 TIMES OLD EXEMPT IS LESS THAN TEMPGROSS
    SET TEMPTAX TO .18 TIMES (TEMPGROSS MINUS 13 TIMES OLD
    EXEMPT).
TAX.
    IF 13 TIMES OLD EXEMPT IS LESS THAN TEMPGROSS SET
    TEMPTAX TO .18 TIMES (TEMPGROSS MINUS 13 TIMES OLD
    EXEMPT).
```

SOURCE PROGRAM STATEMENT FORM

Honeywell

Electronic Data Processing

Title . . . . . Revision . . . . .

Prepared By . . . . . For Program . . . . .

Date . . . . . Checked By . . . . .

Remarks . . . . .

FACT

	P	SERIAL NUMBER	PROGRAM STATEMENTS
1	P		
2	P		
3	P		
4	P		
5	P		
6	P		
7	P		
8	P		
9	P		
10	P		
11	P		
12	P		
13	P		
14	P		
15	P		
16	P		
17	P		
18	P		
19	P		
20	P		

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

FORM NO. T1205

Figure 3.

SUB-PARAGRAPHS: Sub-paragraphs are logical sub-divisions of a paragraph and are indented any desired amount with respect to the master paragraph. (In other words, if the first line is indented n spaces with regard to the first line of the master paragraph, then the body is indented n spaces with respect to the body of the master paragraph.) The name of a paragraph encompasses all included sub-paragraphs. If sub-paragraphs of different master paragraphs have the same name, a specific sub-paragraph reference may be made distinct by using either adjectival or phrasal modification. A blank line must follow a sub-paragraph which is followed by a paragraph starting farther to the left.

SUBROUTINES: A paragraph or sub-paragraph may be used by the programmer as a subroutine. The dynamic abbreviation technique described on page 15 provides a method of generalizing subroutine parameters. For example, paragraph A is a subroutine which computes the interest (I) on a quantity Q at rate R. The same program contains two paragraphs called MORTGAGE and BANKLOAN which execute this subroutine for different values of Q and R. These two paragraphs are written as follows:

MORTGAGE. PUT MASTERFILE-PRINCIPAL (Q) INTO SUMMARY--PRINCIPAL AND MASTERFILE-RATE (R) INTO SUMMARY-RATE. DO PARAGRAPH A. ADD I TO PRINCIPAL.

BANKLOAN. ADD LOAN (Q) TO TOTAL-LOAN. PUT CHARGES (R) INTO CURRENT-RATE. DO PARAGRAPH A. ADD I TO TOTAL-INTEREST.

When A is executed from paragraph MORTGAGE, the subroutine computes interest on MASTERFILE-PRINCIPAL at the rate MASTERFILE-RATE. When A is executed from paragraph BANKLOAN, the subroutine computes interest on LOAN at the rate CHARGES. The fields MASTERFILE-PRINCIPAL and LOAN may be of different lengths.

Procedures

A procedure is a paragraph or group of successive paragraphs which perform a particular computational task. The beginning of a procedure is indicated by either of the titular forms:

PROCEDURE_____.

or _____ PROCEDURE.

If a procedure is assigned multiple names, parentheses are used as follows:

ALPHA (BETA) PROCEDURE.

Then either of the names ALPHA or BETA may be used to reference the procedure.

All source statements following a procedure name are assumed to be a part of that procedure until the programmer writes either the name of another procedure or one of the following statements:

END OF (PROCEDURE)_____.

or END OF _____(PROCEDURE).

Notes

A sentence wholly enclosed in parentheses or a paragraph headed by the name "NOTES." is defined as a note or set of notes. These may be inserted at any point in the source language to describe a situation or a procedure in greater detail. Notes are not compiled and result in no object coding, but are retained with the program and reproduced as part of any program listing. Hence they are not limited as to vocabulary. A paragraph entitled "NOTES." must be followed by a blank line.

Data Placement

The verbs REPLACE and PUT are provided for moving data internally. These verbs are followed by the prepositions INTO and BY, respectively. Either of the following statements will store the contents of field A in field B:

PUT A INTO B.

or REPLACE B BY A.

If the object field is compound, the contents of the subject field will be stored in both places, for example:

PUT C INTO A AND B.

or REPLACE A AND B BY C.

If the subject field is compound, the component subfields are juxtaposed in order of appearance, forming an extended field to be placed in the object field. For example, the statement

PUT A AND B INTO C.

or REPLACE C BY A AND B .,

where A is a 3-character field, B is a 2-character field and C is a 5-character field, will result in C containing as its first three characters the contents of A, and as its fourth and fifth characters the contents of B.

The object program can distinguish blank fields from fields which contain information (even zeros). The programmer may test for blank fields as in the following examples:

PUT A INTO B UNLESS BLANK
IF BLANK, REPLACE B BY A. OTHERWISE, _____

When a Hollerith field is moved internally by a PUT or a REPLACE statement, if the object field is not the same length as the subject field, the information is justified to the left and either truncated or followed by the necessary number of blanks. If the field moved is numeric, its resulting position depends upon the scaling specified on the File Outline. The field is shifted to line up the specified decimal-point positions. The high-order end of the field is either truncated or filled in with zeros. The low-order end of the field is either truncated or rounded (as specified on the File Outline) or filled in with zeros.

The verbs PUT and REPLACE may be used to move either fields or groups. If a group is moved, all included primary subgroups of A, down to but not including the first secondary subgroup, are moved into like-named subgroups of B (if such exist). A special case is the statement

PUT A FIELDS INTO B FIELDS

or REPLACE B FIELDS BY A FIELDS

where A and B are groups. In this case, fields and subgroups of A are placed in B in the order specified by the File Outline, regardless of like or unlike names. Each field moved is treated just as in the case of moving a single field with regard to shifting, truncating, rounding, or filling in blanks or zeros.

If most, but not all, of a group or entry is to be moved, the data to be moved may be specified in a single statement, rather than writing individual statements for each unit of such data. Thus, if a master file group consisting of Name, Street, City, State, Telno, Hours, and Rate fields is to be replaced by a detail file group consisting of the same fields, the programmer may write:

REPLACE MASTER Group Name BY DETAIL Group Name.

In the above example, if all fields except Hours are to be replaced, the condensed statement:

REPLACE MASTER FIELDS BY DETAIL FIELDS EXCEPT HOURS

has the same effect as the lengthier statement:

REPLACE NAME, STREET, CITY, STATE, TELNO, AND RATE OF MASTER BY NAME, STREET, CITY, STATE, TELNO, AND RATE OF DETAIL.

An attempt to move a field of one mode into a field of different mode will result in an error indication during compilation.

Data Deletion

The verb DELETE may be used to remove a field or group from the entry being processed or to remove an entire entry from the file. It has no effect on a field or group not yet created.

DELETE CHANGE - EO (GROUP).

Imperative Arithmetic

Numeric quantities may be combined and the result stored in a different field by the use of the verb SET, as in the statement:

SET PRICE (EQUAL) TO COST PLUS PROFIT

which adds the value of COST to that of PROFIT and stores the result in the PRICE field.

In addition, imperative arithmetic statements may be written using the verbs ADD, SUBTRACT, MULTIPLY, and DIVIDE, as follows:

ADD A TO B.

SUBTRACT A FROM B.

MULTIPLY B BY A.

DIVIDE B BY A. or DIVIDE A INTO B.

Each of these statements produces object coding which performs the stated operation and stores the result as the new value of field B, thus destroying the old value. In any of the above statements, the quantity A may be a field name, a numeric literal, or a numeric quantity (as defined above); however, B must always be a field name. Thus, each of the following is a legitimate imperative arithmetic statement:

MULTIPLY C BY 2.	(2*C ⟶ C)
ADD A TIMES B TO F.	(A*B+F ⟶ F)
DIVIDE X BY Y PLUS 4.	(X/(Y+4) ⟶ X)
SUBTRACT X OVER B FROM E.	(E-X/B ⟶ E)

In any of the basic imperative arithmetic formats above, either the A or the B quantity or both may be compounded by the connective AND. Again, any quantities which are compounded in the B field must be field names. For example:

ADD A AND B TO C.

produces the same result as the two statements:

ADD A TO C. ADD B TO C.

Similarly,

ADD A TO B AND C.

produces the same result as:

ADD A TO B. ADD A TO C.

As a slightly more complicated example:

SUBTRACT A AND B FROM C AND D.

has the effect of:

SUBTRACT A FROM C. SUBTRACT B FROM C.
SUBTRACT A FROM D. SUBTRACT B FROM D.

A series of simple and/or compound arithmetic imperatives may be written as a single statement, using either the connective AND or a comma, as follows:

ADD A TO B, SUBTRACT C FROM D AND E AND DIVIDE E BY F.

Validity

If a Hollerith field of one maximum size is put in another Hollerith field of a smaller maximum size, the overflow characters on the right will be lost and a validity error will occur. The same problem arises with decimal fields in other ways. Many arithmetic operations may create numbers which contain more digits than are provided for in the object field. If the excess digits are to the right of the low-order end of the object field, they are truncated or rounded as specified by the object field description. (In this case, there is no validity error.) If the excess digits are to the left of the high-order end of the object field, the number is too large for the field.

If A is a numeric field, too large for B, the statement

PUT A INTO B

results in a validity error. Similarly, such errors may occur as the result of arithmetic operations.

Two forms may be used in a statement to check the operation for validity error and take corrective action if necessary. In the first case, the statement:

IF VALID, PUT A INTO B, OTHERWISE DO ERROR.

puts A into B unless a validity error occurs. However, if A is too large for B, nothing is put into B and the procedure named ERROR is executed. Thus, this statement results in an either/or situation. The second form is written:

PUT A INTO B AND IF INVALID DO ERROR.

This statement again puts A into B if no validity error occurs. However, if A is too large for B, it puts a truncated version of A into B and then executes the error procedure.

Certain arithmetic statements imply the creation of many intermediate quantities, of which any one or more might be invalid. For example, the statement:

SET C EQUAL TO (A + B) / (A - B) * (A - B * B).

results in a sequence of six operations which may produce one or more validity errors. On the other hand, the statement:

IF VALID SET C EQUAL TO (A + B) / (A - B) * (A - B * B),
OTHERWISE DO ERROR.

causes the error procedure to be executed if any of the intermediate results is invalid. The expressions IF VALID or IF INVALID apply only to the sentence in which they appear.

A validity error which occurs as the result of a statement containing no IF VALID or IF INVALID provision is called an unchecked validity error. The occurrence of such an error may subsequently be checked by writing an expression of the form:

IF NO UNCHECKED VALIDITY ERROR,
or IF UNCHECKED VALIDITY ERROR.

Statements of the second type may be followed by an OTHERWISE provision. Either of these statements checks all previous unchecked validity errors since the start of the paragraph in which it appears. If validity errors occur because of a paragraph or procedure which is inserted by means of a SEE, DO, or EXECUTE statement (see page 26), such errors are not checked unless the inserted matter contains its own validity checks. However, validity checks can be made on one or more specific paragraphs or procedures by writing a statement which includes the condition:

IF NO UNCHECKED VALIDITY ERROR IN A, B, AND C,.

Where a portion of a program involves multiple possibilities for validity errors, programming is simplified by checking entire paragraphs for validity rather than checking each operation individually.

Control Statements

Three types of control statements may be used to perform a portion of the program other than the next sequential portion. The first type, designated by any of the verbs SEE, EXECUTE, or DO, directs the program to perform a single specified paragraph or procedure and then return to the statement following the control change.

SEE ALPHA.
DO NEW-ACCOUNT PROCEDURE.

The verbs SEE, DO and EXECUTE are treated identically by the Compiler. They may also be used to change control to a sequential group of paragraphs or procedures before returning to the original coding by writing

DO 11 THROUGH 14.

where 11 and 14 are assigned as paragraph numbers. The still more extended usage

DO 11, 12, 15 THROUGH 21, AND 24.

is allowed. (The alternate spelling THRU is also acceptable to the Compiler.)

The second type of control statement, which directs the Compiler to change control to a specified paragraph or procedure without returning to the original coding, may be designated by any of the following forms:

GO TO _____.

or SKIP TO _____.

or RETURN TO _____.

The third type of sequence-changing control statement is:

LEAVE PROCEDURE.

or LEAVE PARAGRAPH.

A statement of this form may be inserted in a procedure or paragraph, usually in connection with a condition. When executed, it causes a control change to the next operation to be performed following the procedure. That is, if the procedure was reached in normal sequence or by a SKIP, GO, or RETURN statement, LEAVE PROCEDURE changes control to the next sequential operation. However, if the procedure was reached by a SEE, DO, or EXECUTE statement, LEAVE PROCEDURE changes control to the operation following the SEE, DO, or EXECUTE.

Conditional Statements

Several methods may be used to write conditional statements. The operation to be performed may be made dependent upon the relative magnitude of two quantities by writing one of the following expressions:

IF X IS GREATER THAN Y,

or IF X IS LESS THAN Y,

or IF X IS (EQUAL TO) Y,

or IF X EQUALS Y.

As stated earlier, the words in parentheses may be omitted without change of meaning. Any of the above expressions may be written using the word WHEN in place of IF. A conditional clause may be made negative by the inclusion of the word

NOT (IF X IS NOT GREATER THAN Y, etc.) or by using the word UNLESS in place of IF in the above forms.

A condition affects only the statement in which it appears. If the condition is followed immediately by the word OTHERWISE, then an either-or situation is established. In this case, if the condition is met, all statements are performed up to the OTHERWISE, after which control skips to the following sentence. If the condition is not met, the conditional statement is not performed and control skips to the OTHERWISE. A simple example follows:

IF X IS (EQUAL TO) Y, ADD A TO B. OTHERWISE SUBTRACT A FROM B.

Use of Tape Files

Except where using the powerful functions SORT or UPDATE which implicitly include all necessary file handling procedures, a tape file must be manipulated by explicit program statements.

OPENING FILES: Before any tape file may be used, one of the following statements must be written:

OPEN (FILE) A.

or GET (FILE) A.

More than one file may be opened by normal compounding of these statements, as in the example:

OPEN (FILES) A, B, AND C.

In addition to opening the file, the OPEN statement reads all primary information at the file level into memory until a secondary group is encountered. Such information might include, for example, file name, number of entries, date, and other information not related to specific entries. The GET statement also opens the file and reads the same information into memory. In addition, it reads one of each secondary group into memory until a secondary group is encountered whose rank is equal to or higher than that of its predecessor. The amount of information read by a GET FILE statement is called the first information hierarchy of the file.

Before an output file can be created, one of the following statements must be written:

OPEN NEW (FILE) A.

or FILE (NEW) (FILE) A.

The verbs OPEN, FILE, and GET when applied to files cause the necessary reading or writing of labels and directories, and any other action which is required prior to obtaining or preparing the first records of a file.

CLOSING FILES: When processing for a tape file is complete, the file is closed by the statement:

CLOSE (FILE) A.

All required action by the computer, such as the recording of trailing labels, is performed by object coding produced by the CLOSE statement. Every file opened must be closed. The CLOSE statement may be compounded in the same manner illustrated for the OPEN statement.

GETTING AND FILING INFORMATION: Once a file has been opened, groups of information may be obtained from it by means of OPEN or GET statements and recorded by means of FILE statements. Here, as elsewhere, the term group includes entry, which is the highest ranking group under a file. The statement:

GET NEXT group name.

obtains the first hierarchy of information in the next group of the given name. The programmer can use OPEN in place of GET in any statement described here. In each case, an OPEN statement obtains only the primary information of the top level of the group instead of the entire first hierarchy as with the GET statement. If the group contains various subgroups, any lower ranking group obtained may be determined by the condition:

IF group name.

To obtain a specific type of group within the current next higher ranking group, the programmer may write:

GET group name.

and may include in the same sentence a condition:

(OR) IF NONE

followed by the procedure to be performed if this group is not present. If the programmer requests a group which does not occur in the current next higher ranking group, or, if he does not indicate the above conditional clause and no group is found, then an error is signaled and diagnostic information is produced.

If the programmer writes:

GET NEXT GROUP.

the program obtains the next information from the file. This may be the first hier-

archy of a new entry or the next hierarchy of the current group. In order to skip over intervening data to obtain another group of specific rank, the programmer may write:

GET NEXT GROUP WITHIN group name.

A test for the end of the current group, entry, or file may be performed by writing the following conditional phrase:

IF END OF group name, .

Information may be recorded in a file by writing the following statement:

FILE group name.

This statement files all primary information in the specified group down to the first included secondary group.

After filing information on tape, the programmer may wish to indicate that the end of a higher-level group has been reached and that no more information is to be filed in that group. This is accomplished by writing the statement:

CLOSE (group name).

A CLOSE statement closes the specified group and all included open groups. A subsequent FILE statement will file all closed higher-level groups, thus starting a new higher-level group on tape, as well as filing the specified information.

Two precautions are pointed out regarding the obtaining and the recording of file information. If data is generated in a secondary group before the preceding secondary group of the same level is filed, data in the preceding secondary group is lost. Also, if a request for a specific group name is not accompanied by a test for the end of a higher-level group, large amounts of data may be passed over before a group of the specified type is located.

In order to transfer an entire group or the balance of a group from one file to another, the programmer may write:

FILE ENTIRE (GROUP) (group name) FROM (FILE) file name.

The two files involved must have identical file outlines. The programmer must have given a prior OPEN or GET statement to obtain at least part of the group. He may have changed and/or already filed any parts of the entry or he may have inserted a new group by means of a FILE statement. However, any parts of the entry which have been passed over (not filed and no longer available) will be lost. When a FILE ENTIRE statement is given, it is not possible for the object program to tell whether

the last output filed was from a group just obtained or from a previous similar group. In order to allow the programmer to file changes or additions prior to a FILE ENTIRE statement, the program always assumes that as much information has been filed as possible in view of the last type of group filed. If the information last filed was from a previous group, the programmer should close that group subsequent to filing it and prior to the FILE ENTIRE GROUP statement. The phrase FROM (FILE) file name may be omitted if no ambiguity is possible. On the other hand, any FILE or FILE ENTIRE statement may have a phrase

IN (FILE) file name

if such is needed to remove ambiguity.

REVERSE FILE: If the programmer writes

REVERSE (FILE) file name.

Succeeding GET and OPEN verbs are executed with the tape moving backward until a second REVERSE FILE statement sets up forward reading. The verb REVERSE should only be applied to reversible files. The verbs DELETE and FILE may not be applied to a file which is reversed.

RESTART: To facilitate restart procedures, the statement

SET RESTART

can be inserted at those points in the program where restart breaks are desired.

Definitions

A definition is an expression containing no verbs except "is" or "equals" or their variants. Each definition must be assigned a name and made a paragraph. Whenever a definition is referenced in a conditional statement, the definition is tested to determine the resulting action. For example, the name "SENIOR" might be defined as follows:

SENIOR. TENURE IS GREATER THAN 10 AND RANK
IS GREATER THAN LEADMAN.

This definition can then be used as the basis of a conditional statement of the type:

IF SENIOR ADD BONUS TO PAY.

The object program will examine the fields called "TENURE" and "RANK" in each group to determine whether or not the condition is met.

Definitions may be combined and a name assigned to the combination. For example:

ELIGIBLE. SENIOR AND GRADUATE OR OFFICER.

Here, the names "SENIOR", "GRADUATE", and "OFFICER" must be individually defined. (Note that ANDs are performed before ORs. Parentheses are used to denote any exception to this distribution.) In this case, the eligibility of employees (as for a training course) is determined either by seniority plus education level or by company officer status. The connectives which may be used in such combined definitions are AND, EITHER, OR, NEITHER, and NOR. The same connectives may be used to combine conditions within a conditional sentence, with parentheses included if necessary to clarify their distribution. It is important to note that definitions produce no results by themselves, but only specify the manner in which a test is to be performed.

Another method of indicating a conditional situation is by means of a value or a set or range of values of a particular field, called a conditional field. The value or range of the conditional field which satisfies the condition can be used as an adjective. For example, a payroll file contains a field called NAME and a conditional field called SCHEDULE in a group called EMPLOYEE. The latter may have either of two values: "S" for salaried or "H" for hourly. To perform a particular operation only on hourly employee records, the statement:

PUT NAME OF HOURLY EMPLOYEE INTO ADDRESSEE.

specifies in adjectival form the same condition expressed in the longer statement:

IF SCHEDULE OF EMPLOYEE IS (EQUAL TO) ..H.., THEN PUT NAME OF EMPLOYEE INTO ADDRESSEE.

Either statement results in a test of the value of "SCHEDULE" to determine whether or not the stated action is to be performed. The programmer could equally well have said:

IF HOURLY EMPLOYEE, THEN--------

LOGICAL MULTIPLIER: A logical multiplier is a factor in an arithmetic expression or statement which can have either of the values one or zero. The programmer may use the name of a definition as a logical multiplier. When the arithmetic is performed, if the definition is true, its name has the value one; otherwise its value is zero. For example, the following definition might be used as a logical multiplier:

OVERTIME. HOURS IS GREATER THAN 40.

by referencing its name in the arithmetic statement:

SET GROSSPAY TO RATE * (HOURS + .5 * OVERTIME * (HOURS-40)).

When this statement is performed, if the definition of OVERTIME is true, this word is assigned the value one; otherwise it is assigned the value zero. This is equiva-

lent to writing the long conditional statements:

IF HOURS IS GREATER THAN 40, SET GROSSPAY TO RATE * (HOURS + .5 * (HOURS - 40). OTHERWISE SET GROSSPAY TO RATE * HOURS.

Sorting

The SORT statement is an especially powerful feature of FACT which causes the generation of a merge-sort routine specifically tailored to perform the requested sorting with the amount of high-speed storage and the number of magnetic tape units provided. Unless a sort is small enough to be performed entirely internally, a minimum of three tape units is required. When worthwhile, the first phase of the generated merge-sort routine will automatically be processed in parallel with the preceding source statements in order to save the time required for one pass over the input data. Likewise, when worthwhile, the last phase of the generated routine will automatically be combined with the following source statements in order to accomplish some additional work while the sorted data is being written on tape.

The simplest form of the SORT statement is:

SORT (file name).

This statement sorts every group in the specified file separately and according to all of its component fields. The relative significance of these fields as sort keys is determined by their order in the file outline. The sorted groups are also ordered as specified by the file outline. This type of sort is illustrated in terms of the simple file structure given in Figure 4.

```
* A
      B
    * C
          D
        * E
              F
              G
    * H
          I
          D
```

Figure 4.

If this file is sorted by writing the above statement, the sort routine first sorts all E groups within each A group, using F and G as major and minor keys, respectively. All C groups are then sorted on field D, carrying the ordered E groups with them. The H groups are sorted on I and D and finally, the A groups are sorted on B. Within A, all of the ordered C groups end up ahead of all ordered H groups, as specified by the file outline.

The most general SORT statement takes the form:

SORT (group name) ON (field names) WITHIN (higher-ranking group name).

This statement sorts the specified group according to a key made up of the fields named. File keys must be named in order of significance, from major to minor. Groups having identical keys remain in the order that they had in the original file. Groups not specified in the SORT statement follow the sorted groups in the order that they had in the original file, but all groups within the specified higher-ranking group remain within that group. All groups of higher rank than the sorted groups remain in their original order. Referring to the file shown in Figure 4, the statement:

SORT E ON G AND F WITHIN C.

causes the sorting of all E groups on field G and then on field F within their related C groups. All C, H, and A groups retain their original order and all C groups precede all H groups within any A group.

The SORT statement may be compounded so that different groups are sorted on different keys, as in the statement:

SORT E ON G AND F WITHIN C AND H ON D AND I WITHIN A.

This statement causes sorting of E groups within C and H groups within A, but leaves C and A groups in their original order, as well as all C groups ahead of H groups within the related A group.

If only one set of keys is specified for two or more kinds of groups, the specified groups are intermixed and sorted jointly according to the stated keys. All unspecified groups will follow the ordered groups and retain their original order. This case is illustrated by the statement:

SORT C AND H ON D WITHIN A.

Referring again to Figure 4, this statement intermixes C and H groups and sorts them all together on the common key D. All E groups retain their original order within the related C groups.

If the WITHIN phrase is omitted, FACT assumes that the phrase WITHIN FILE

is intended. In this case, if the specified group appears in several different higher-ranking groups, these higher-ranking groups are intermixed and sorted jointly. For example, the statement:

SORT E ON F.

intermixes all E groups from all C groups and sorts them all jointly. Since sorting must not remove any information from its proper higher-level group, this type of sort will necessitate duplicating much higher-level information in the sorted file. Figure 5 shows a specific portion of a file organized according to Figure 4, both before and after sorting by the above statement. Note that after sorting two C groups in this example, four C groups are recorded in the sorted file. In Figure 5, the numbers in parentheses represent the values of the keys F.

Before Sorting

```
C1
    D1
    E11
        F11 (1)
        G11
    E12
        F12 (7)
        G12
C2
    D2
    E21
        F21 (2)
        G21
    E22
        F22 (8)
        G22
    E23
        F23 (9)
        G23
```

After Sorting

```
C1
    D1
    E11
        F11 (1)
        G11
C2
    D2
    E21
        F21 (2)
        G21
C1
    D1
    E12
        F12 (7)
        G12
C2
    D2
    E22
        F22 (8)
        G22
    E23
        F23 (9)
        G23
```

Figure 5

If (group name) is omitted, the result is the same as though a separate SORT statement were written for every group containing the specified fields. In this case, each such group is sorted separately and left in the original group sequence. For example, the statement:

SORT (file name) ON (field names).

causes sorting of all groups in the file outline which contain any of the specified fields. All groups of the same type are sorted separately on these fields and the final sequence of the ordered groups is as specified on the file outline.

If a numeric key (or keys) is specified, the data will be sorted to arithmetic order (negatives less than positives). If a non-numeric key (or keys) is specified, the data will be sorted to standard collator sequence (blanks, punctuation marks, alphabetic characters, and numerals). In either case, the output will be in normal ascending sequence unless the verb SORT is preceded by the word REVERSE.

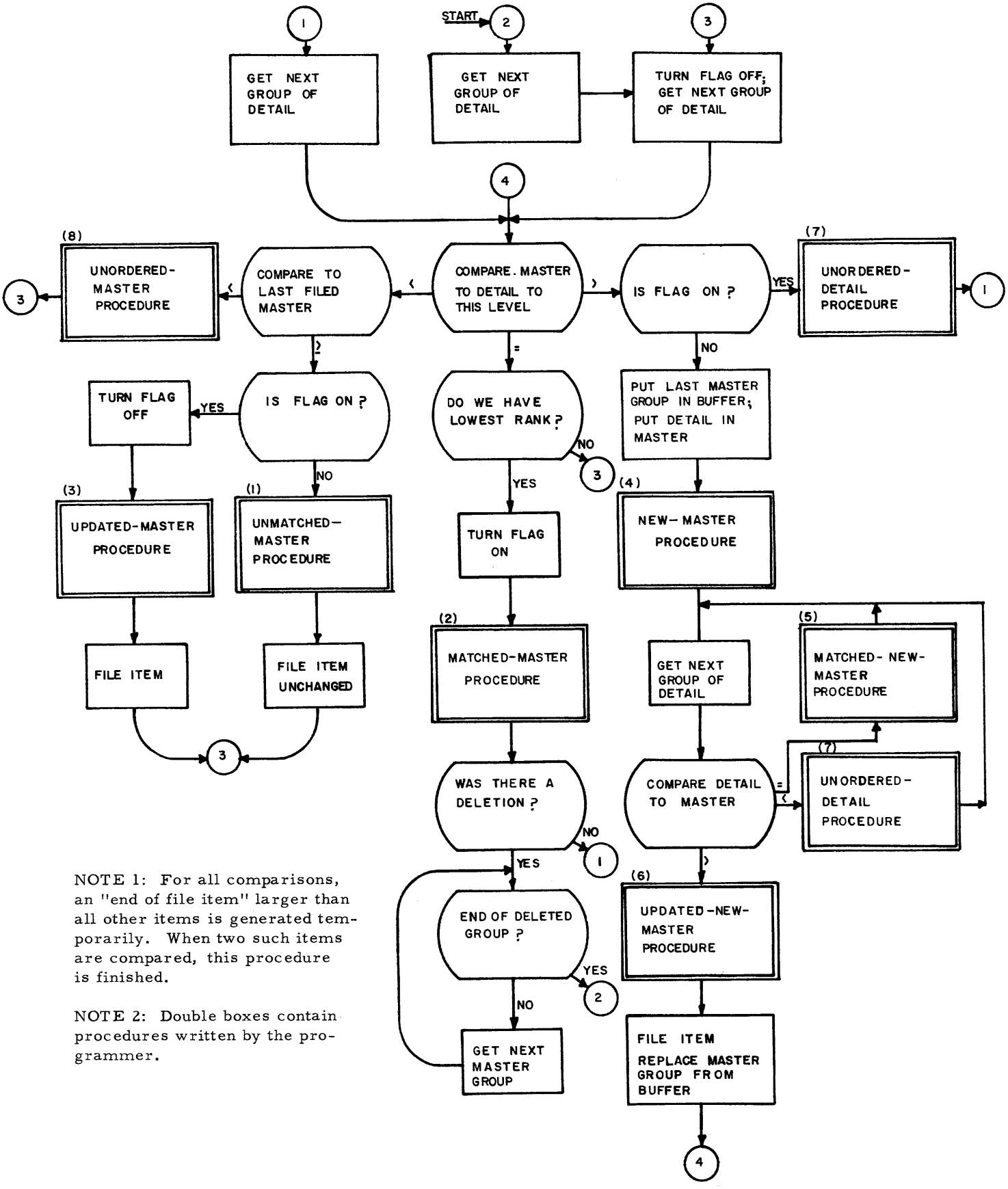
The Update Function

Updating is the process of modifying (usually periodically) the information in a permanent master file according to the transactions and other activity in a detail file. This process results in a new updated master file tape and, in many cases, in a variety of reports and other output. One of the special features of FACT is the ability to generate updating routines for files containing entries of a single type.

The updating routines generated by FACT perform the following operations:

(1) GET information from master and detail files, compare the keys of master and detail file entries, and test for the end of the data;
(2) UPDATE a master entry each time it is matched by a detail entry;
(3) Perform any necessary procedures each time that the compared keys do not match;
(4) Perform any necessary procedures each time a master or detail entry fails to occur in the correct sequence; and
(5) FILE updated information in new master file.

The UPDATE statement makes a generalized or skeletal updating routine available to the object program. Figure 6 is a flow chart of this skeletal routine, which contains essentially the coding to perform operations (1) and (5) above. This routine is specialized to perform a particular update by the statement of a series of procedures which the programmer provides to perform the other three operations above. Referring to Figure 6, these procedures are represented by eight procedure boxes



NOTE 1: For all comparisons, an "end of file item" larger than all other items is generated temporarily. When two such items are compared, this procedure is finished.

NOTE 2: Double boxes contain procedures written by the programmer.

Figure 6. Flow Diagram of Routine Generated by the UPDATE Statement

identified by the arabic numerals from 1 to 8. The successful interpretation of the UPDATE statement requires that the programmer furnish properly titled procedures to handle these eight circumstances. The successful operation of the generated routine requires that the master and detail files be arranged in the same sequence and that every entry in the master file be identified by a unique key. However, detail entries may have duplicate keys and entries may appear in either file which do not appear in the other.

The following are the procedures that are used by FACT to generate an operating update routine. These procedures must be given the specified names in the source program.

(1) UNMATCHED-MASTER PROCEDURE. This procedure is executed when a master entry is not matched by a detail entry. If this procedure is not specified, the updating routine writes the unmatched master entry on the new master file and reads another master entry from the old file. A detail entry with a higher key than the unmatched master entry remains in memory for the next comparison.

(2) MATCHED-MASTER PROCEDURE. When a master entry and a detail entry are matched, the routine executes this procedure, which includes the source statements required to perform the desired updating. If the detail entry deletes the master entry, new entries will be read from both files.

(3) UPDATED-MASTER PROCEDURE. This procedure is executed when a master entry has been updated by all matching detail entries and is ready to be written on the new master file tape.

(4) NEW-MASTER PROCEDURE. When a detail entry does not match any master entry, this procedure is executed to create a new master entry out of all detail fields which have the same names as master fields.

(5) MATCHED-NEW-MASTER PROCEDURE. When a new master entry created by procedure (4) above is matched by a detail entry, the new master entry is updated in accordance with the source statements of this procedure. Thus a series of unmatched detail entries having the same key will create a new master entry and then update it.

(6) UPDATED-NEW-MASTER PROCEDURE. This procedure is executed when a new master entry created by procedure (4) above has been updated by all matching detail entries according to procedure (5) above

and is ready to be written on the new master file tape.

(7) UNORDERED-DETAIL PROCEDURE. This procedure is executed when a detail entry is out of sequence. A new detail entry is read.

(8) UNORDERED-MASTER PROCEDURE. This procedure is executed when a master entry is out of sequence. A new master entry is read.

In summary, the desired updating routine is generated from a statement of the form:

UPDATE B BY A. CONTROL ON K_n, K_{n-1}, ...K_2, AND K_1.

together with the necessary procedures which tailor the skeletal routine to the particular application. B represents the name of the old master file; A the name of the detail file; and K_n through K_1 the keys to be used in matching entries, in the order from higher to lower levels in the file hierarchy. The UPDATE statement is immediately followed by a series of procedures having the titles listed above. If the programmer wishes to execute the same procedure in two or more of the situations represented in the flow chart, he may assign multiple titles to this procedure using parentheses. For example, if the same procedure is used to update a matched master and to update a matched new master, this procedure may be titled:

MATCHED-MASTER, (MATCHED-NEW-MASTER) PROCEDURE.

The use of the update function is illustrated in the sample problem described in Section VI.

Report Preparation

The third major function specifically built into FACT is the preparation of printed reports from magnetic tape files. The process by which such reports are described as input to FACT is described in detail in Section IV. Once a report has been unambiguously described by means of this process, a statement of the form:

WRITE (REPORT)_____.

will cause the information pertaining to this report from the current entry to be recorded on a report tape. If any ambiguity exists as to the location of information comprising the report, the WRITE statement can be amplified by means of the word FROM followed by the appropriate group, entry, or file name.

Information for printing a number of different reports can be interspersed on the report tape for selective printing. Alternatively, report information can be reproduced directly as it is generated. As in the case of the sorting and updating functions, the necessary information access statements (OPEN, CLOSE, FILE, and GET) are

automatically provided by the report generator. If the programmer should desire to begin the next information on a fresh page, he should use the statement:

CLOSE PAGE (OF REPORT _____).

Tables

The programmer may include in his source statements one or more tables of information (such as prices, taxes, freight rates, or trigonometric functions) to be used by the object program in the course of some computation. Other source statements can then look up any information presented in tabular form for use as operands.

The beginning of a table is indicated by either of the following titles:

TABLE _____.

or _____ TABLE.

Following the title, the names of the respondents are listed in standard series form and terminated by the word OF and the name of the argument, as follows:

BOND TABLE. Q-BOND-VALUE, E-BOND-VALUE, AND
H-BOND-VALUE OF YEAR.

Each respondent and the argument must be defined as a field in an Internal File Outline.

Following the title and description of the table, the columnar positions of the argument and the respondents are laid out on the Source Program Statement Form by writing their names in the appropriate columns. The name of the argument must be written to the left of the respondent names. The columnar position of each name indicates the approximate columns of the source document in which the values of that quantity will appear. This line is followed by the necessary number of lines comprising the table. The values listed for each quantity should be left justified within the columns so defined for non-numeric quantities and right justified within these columns for numeric quantities. The mode of these quantities is interpreted by FACT from the Internal File Outline. Each data column in the table must be separated from the adjoining column(s) by at least one space.

If the table contains more respondents than can be listed on a single line of the Source Program Statement Form, the table may be divided into sections. All values of the argument and the corresponding values of the first-section respondents are then written in the first section of the table. The beginning of a new section is then indicated by one of the following statements:

TABLE _____, CONTINUED.

or _____ TABLE, CONTINUED.

The continuation statement is then followed by a line of column headings as before, with the name of the argument at the left and the names of the second-section respondents in the appropriate columns. A table may be broken into as many sections as necessary in this manner. Each section must contain all of the same values of the argument and all of the corresponding values of the included respondents. The end of a table is indicated by one of the following statements:

END OF TABLE _____.

or END OF _____TABLE.

Table look-up is accomplished by writing the name of the desired respondent, the word OF, and the quantity to be used as argument. Note that the quantity to be used as argument can be referenced either by its field name or by its literal value.

For example, if a source program contains the bond table referred to above, that same program can reference data from the table by means of a phrase such as:

E-BOND-VALUE OF 1926

using this phrase as an ordinary numeric literal. FACT interprets this phrase by referring to the respondent named E-BOND-VALUE in the bond table and selecting the value of this respondent which corresponds to the argument value 1926. Alternatively, the program can reference the table by means of the statement:

E-BOND-VALUE OF YEAR

where YEAR is the name of a field containing one of the argument values listed in the table. If more than one table in the program includes the respondent E-BOND-VALUE, the desired quantity may be more specifically referenced by the phrase:

E-BOND-VALUE OF 1935 IN BOND TABLE

Lexicon

The basic FACT vocabulary consists of the words listed in Table I, page 93, plus the cardinal numbers (from zero to billion) and ordinal numbers (from first to billionth) and the punctuation marks ( ) , .. " and . No lexicon word may be used with any meaning other than that illustrated in Section III. However, they may be included in hyphenated multi-word names which are defined in the program.

SECTION IV
REPORTS

The requirements of a complete compiler system include the ability to generate printed or punched card reports from information stored in tape files or derived from processing operations. FACT includes a powerful and flexible report generator which is activated by simple source statements. The description of each report to be generated is part of the input to FACT and includes a description of each type of line and each field comprising the report. Once a report has been completely described, each execution of a WRITE REPORT statement will generate the report information pertaining to the file data currently being processed. This information is normally recorded on a special report tape for subsequent on-line or off-line printing or punching. A number of different reports may be intermixed on the report tape for selective printing or punching. If the programmer wishes to use terminal equipment rather than a magnetic tape unit during the processing run, he may designate immediate reporting. In this case, report information will be printed or punched directly as it is generated.

Tabulation

Tabulation is the process of accumulating the individual values of certain detail fields until a "control break" is reached. At that point, a total line is generated containing the accumulated values of the tabulated fields. This process can be extended by accumulating these first-level totals until a different type of control break is reached and then generating first- and second-level total lines. In this manner, tabulation can be performed at many different levels within the same report.

For example, a file which is in sequence by employee number within group and by group number within department might have a first-level control break on a change in group number and a second-level control break on a change in department number. Since employee number is the lowest level, its detail values are printed at level 00. Whenever a change in group number is encountered, a total line at level 01 is generated following the last detail line in the group. Similarly, whenever a change in department number is encountered, total lines at levels 01 and 02 are generated following the last detail line in the department. In general, a control break at any level causes the generation of total lines at that and all lower levels.

Report Description

The complete description of a report consists of:

(1) A Report Name Descriptor;

(2) A pair of Report Line Layout Descriptors for each line of the report (optional);

(3) A Report Line Action Descriptor for each line of the report; and

(4) A Report Field Descriptor for each field of the report.

At the programmer's option, the Line Layout Descriptors may be entirely eliminated and their contents represented on the Field Descriptors. This option depends entirely upon the convenience of the programmer. If Line Layout Descriptors are used, their contents are presented on the Report Layout Form shown in Figure 7, pages 44-45. The contents of the Report Name Descriptor, Report Line Action Descriptors, and the Report Field Descriptors are presented on the Report Description Form shown in Figure 8, page 46. All of these descriptors are prepared in fixed-field format. The following details, which outline the description of printed reports, are equally applicable to punched reports.

Report Name Descriptors

These descriptors are used to assign the report name and form a means of reference to the report from the source program statements. One such descriptor must be prepared for each report to be generated. Report Name Descriptor information is presented on the Report Description Form (Figure 8).

DESCRIPTOR TYPE -- Column 1: A Report Name Descriptor is identified by an "R" punched in column 1.

SERIAL NUMBER -- columns 2-6: The use of serial numbers on the Report Name Descriptor is identical to the use of such numbers on the File Outline Form (see page 8).

LINES-PER-PAGE -- columns 7-9: Up to 999 lines per page, including interline spaces but not including top and bottom margins, may be specified in these columns. This information must be right justified. If these columns are blank, the desired paperfeed controls must be individually specified for each report line on the Report Line Action Descriptors.

REPORT NAME -- columns 10-80: The report name is used to reference the report in the source program statements. As with other names, embedded blanks are not permitted in the report name; hyphens must be used to separate the words in a multi-word name.

Report Line Layout Descriptors

If the programmer wishes to use Line Layout Descriptors, he must prepare two such Descriptors for each line of the report. Each pair presents a 120-column image of a report line (or an 80-column image of an output card). Report Line Layout Descriptors are punched from information written on the Report Layout Form (Figure 7).

DESCRIPTOR TYPE -- column 1: Report Line Layout Descriptors are identified by an "L" punched in column 1.

SERIAL NUMBER -- columns 2-6: The use of serial numbers complies with the standard use of such numbers, except that the second card of each pair contains an X overpunch in column 6.

SUBLINE NUMBER -- column 7: Two or more physical lines of print may be grouped together and described as a single report line (i.e., the number of print positions in a report line may be 120, 240, 360, etc.). A 360-position report line will be "folded" or presented as three physical lines of print. This device is useful in crossfooting applications (see page 52) since the fields which are used as operands and result in a single crossfoot operation must appear in the same report line. Each physical line of print within a folded line is assigned a subline number which is specified in column 7. Subline descriptors may have continuation descriptors.

LINE NAME -- columns 8-25 (1st card only): The name of a line may contain as many as 18 characters. Hyphens must be used in place of embedded blanks in multi-word names. The line name may be referenced directly in source program statements.

LINE IMAGE -- 1st card columns 26-80, 2nd card columns 8-72: This area, containing a total of 120 columns, represents an image of the line layout, complete with field placement symbols and literal information. These columns have a one-to-one correspondence with the 120 print positions which are active during any printing run. The placement of a field in the line layout is indicated by marking a carat (^) at the point where the low-order (rightmost) character of the field is to be printed. Any Hollerith character written in the line image area will be literally reproduced in the resulting printed line, with the exception of periods and commas, which are described below. If the contents of a field extend into print positions where literal characters are specified, the latter are suppressed and the field is printed. Literals may be used to insert signs, names of units, background characters, or other information in the printed line. If the least-significant character of a literal is to be

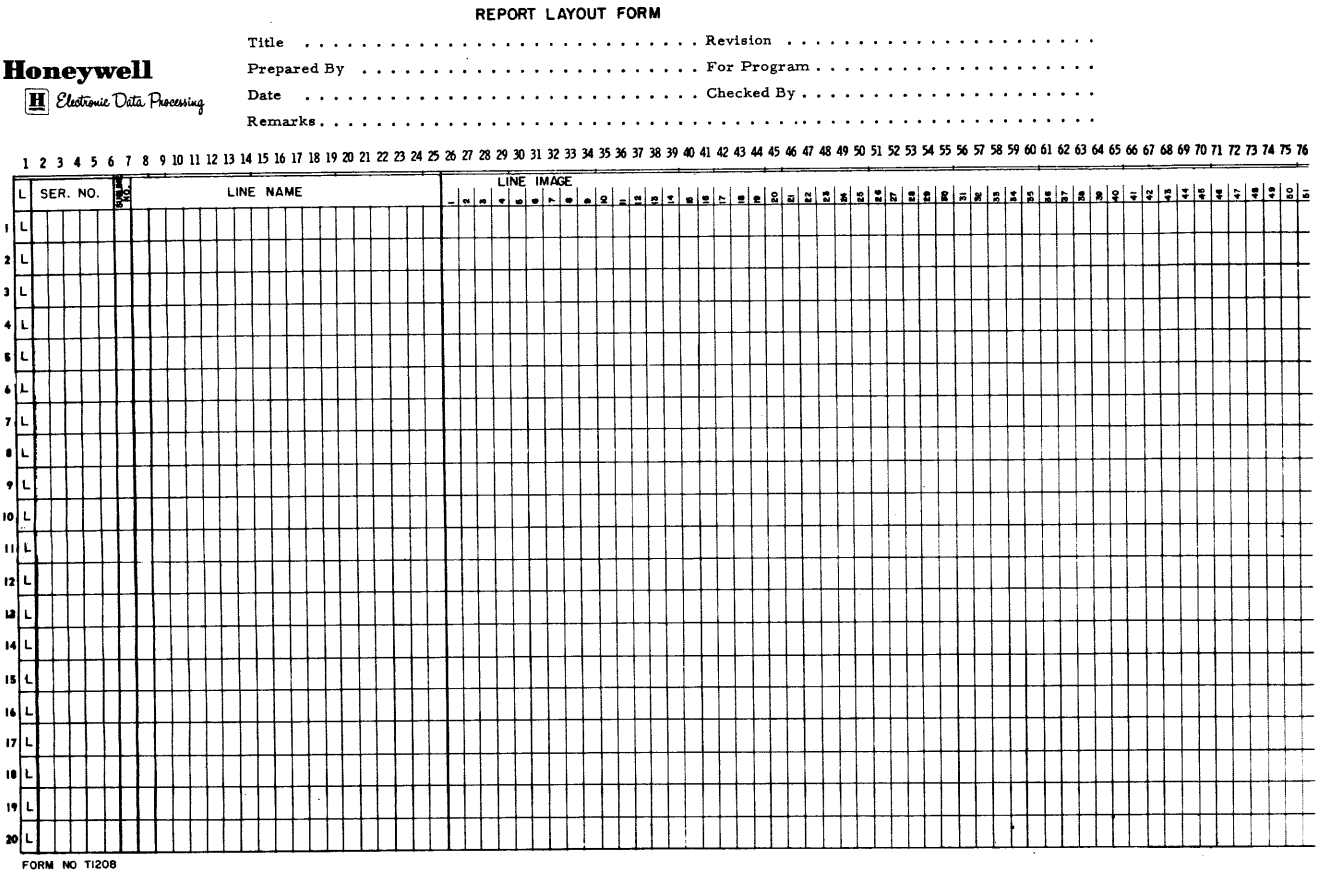


Figure 7.

printed in the right-most position of a field, the carat for that field cannot be placed in the line image. Instead, the programmer must specify the right margin of this field in columns 46-48 of the corresponding Field Descriptor (see page 55).

The fields which comprise a printed line are inserted in the line layout at the print positions indicated by the carats and in the order that they are described on the Report Description Form. All the characters in a Hollerith field are inserted into the line format. In a decimal field, leading zeros are automatically suppressed. A decimal field which is not scaled internally is printed in the indicated print positions unless shifting is specified (see below). A literal decimal point may be written in the line layout to indicate scaling of the printed field. An internally scaled decimal field, on the other hand, is automatically shifted to align the internally carried decimal point with the literal decimal point, except that a specified shift (see below) is in addition to a scaling shift. If no literal decimal point is specified for an internally scaled field, the field is shifted right a sufficient number of places to drop all fractional digits, and the integral digits are rounded and printed. Literal commas may be used to distinguish thousands, millions, billions, and trillions. Such commas will not actually print unless the number of significant digits is sufficient

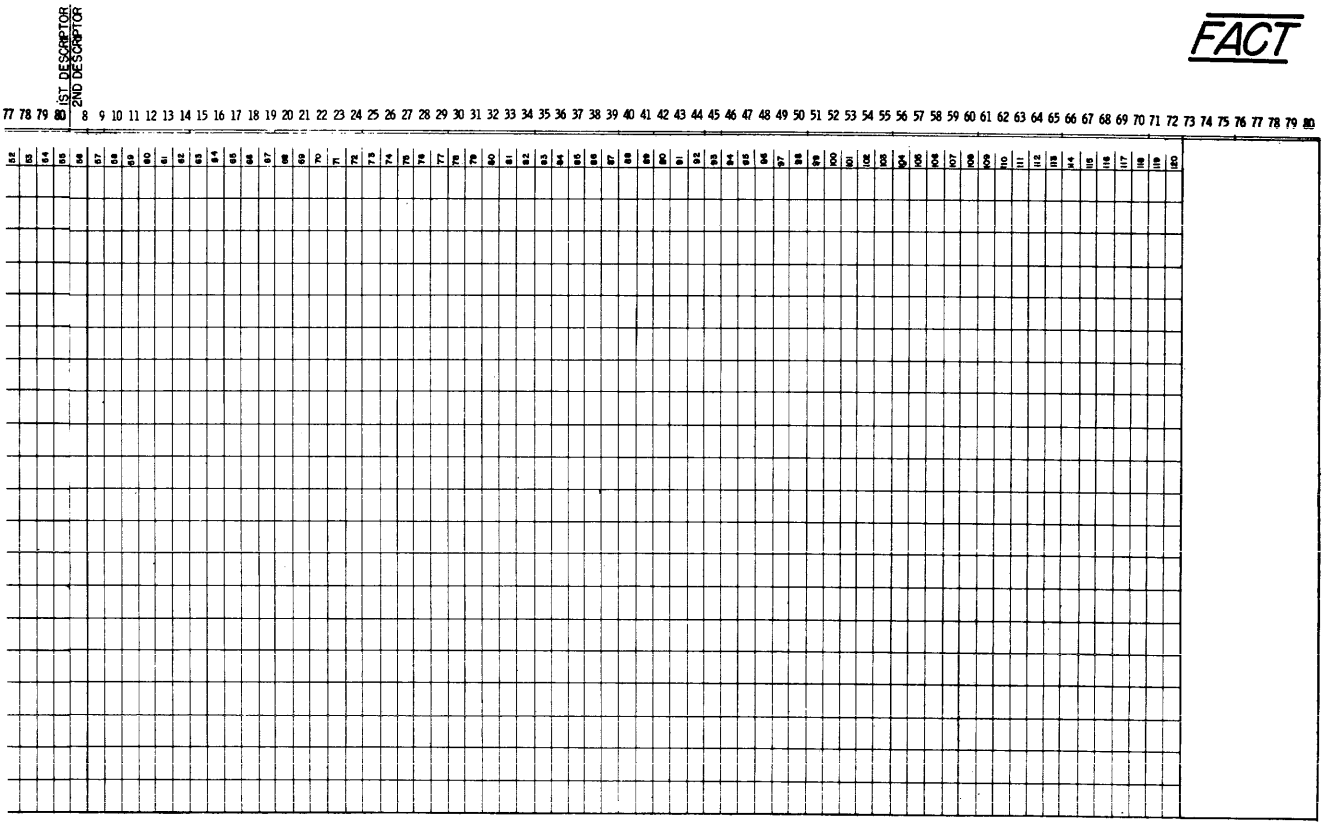


Figure 7 (continued)

to require them.

If the generated report is to be punched, the image of the punched output cards is presented in columns 26-80 of the first card and 8-32 of the second card; columns 33-80 of the second card are not used in this case.

Report Line Action Descriptors

Regardless of the inclusion or omission of Line Layout Descriptors, each line of a report is described by a Report Line Action Descriptor. Lines exceeding 120 print positions ("folded") require a descriptor for each physical print line. These descriptors are prepared using the Report Description Form (Figure 8).

DESCRIPTOR TYPE -- column 1: Report Line Action Descriptors are identified by the presence of an "A" punched in this column.

SERIAL NUMBER -- columns 2-6: The use of serial numbers on the Report Description Form is identical to the use of such numbers on the File Outline Form (see page 8).

LINE NAME -- columns 7-24: If Line Layout Descriptors are used, these columns must contain the same line name as punched on the first card of the corresponding

REPORT DESCRIPTION FORM

Honeywell

H Electronic Data Processing

Title . . . . Revision . . . .
Prepared By . . . . For Program . . . .
Date . . . . Checked By . . . .
Remarks . . . .

FACT

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

R	SER. NO.	LINES PER PAGE	REPORT NAME													
A	SER. NO.	LINE NAME		TYPE OR LEVEL	PRE-PRINT SKIP	POST-PRINT SKIP		TAB CONDITION OR CONTROL FIELD						PRE-PRINT PROCEDURE	PRINT CONDITION	
F	SER. NO	FIELD NAME		TAB ACTION	FIELD SUPP.	CROSS FOOT + − =	Z / G	SIGN	CHECK DECIMAL	INCREMENT	SHIFT TYPE EXTENT	FLOAT CHAR.	MAX. NO. OF CHAR.	LEFT MARGIN	RIGHT MARGIN	FIELD IMAGE
1																
2																
3																
4																
5																
6																
7																
8																
9																
10																
11																
12																
13																
14																
15																
16																
17																
18																
19																
20																

FORM NO. T1207

Figure 8.

pair of Line Layout Descriptors in order to relate the line action information to the corresponding line layout information. If Line Layout Descriptors are not used, these columns are used to assign the line name, according to the standard rules governing names and abbreviations. For folded lines, the line name is omitted on all but the first subline descriptor.

LINE TYPE OR LEVEL -- columns 25-26: A 2-character code is punched in these columns to distinguish among the various types of lines which may be included in a report. The following codes are permitted. Only the first subline of a folded line requires line type specification.

TT (Title Line). Such lines contain identification and descriptive information pertaining to the report. They are composed solely of literal information. The first reference to the report in the source program statements generates all title lines and causes a skip to the top of the next page after these lines are printed. Title lines do not appear again in the same report.

NH (Normal Heading Lines). A report may include one or more of these lines of columnar heading information. All such lines are printed each time a page is ejected under control of the "Lines Per Page" control from the Report Name Descriptor or the "Pre-print Skip" or "Post-print Skip" control (see below). Fields as well as literals may be included.

FL (Footing Lines). A footing line is similar to a heading line except that it is printed at the bottom of each page. Normally, the carriage is directed to eject a page following a footing line. If each page contains a fixed number of lines, the footing line (if any) is the last one printed. If each page contains a variable number of lines, the last line should contain a post-print control (see below); however, a footing line is always spaced to the bottom of the page before the page is ejected. Fields as well as literals may be included.

00 (Detail Lines). These lines, containing information from the data files or created by the object program, form the body of the report. Every source reference to the report generates the entire group of detail lines. To distinguish detail lines from any possible total and final total lines which may be included in a report, the former are referred to by their level (which is defined as 00), rather than by a mnemonic line-type code. A blank line type is interpreted as 00.

01-99 (Total Lines). If a report includes any tabulated fields, the lines which present accumulated values of these fields are called total lines. The Line Action Descriptor for a total line specifies in columns 25-26 the tabulation level, from 01 to 99, to which that line pertains. Total lines do not require line names unless they are referenced in the source program. If a control field occurs in a total line, the preceding detail value of that field is used.

The programmer may designate a pair of total lines for each control break, one containing columnar headings for the subtotals and the other the actual subtotals. Both of these lines will be generated by each control break at the corresponding level. Moreover, either of these total lines may be folded.

FT (Final Total Lines). This line is a variant of the total line which contains final totals of all accumulated fields and is usually printed at the end of the report. It carries an implicit level which is one higher than the highest-level total line printed.

PRE-PRINT SKIP -- columns 27-28: These two columns specify the paper feeding which is to take place prior to printing a line. Each descriptor may specify a separate pre-print skip. Permissible controls are:

EJ - Eject to the first print line of the next page before printing this line;

01-99 - Skip the specified number of lines before printing this line;

Blank or 00 - Do not advance paper before printing this line.

POST-PRINT SKIP -- columns 29-30: These columns indicate the paper feeding which is to take place after printing a line. Each descriptor may specify a separate post-print skip. Permissible controls are:

EJ - Eject to the first print line of the next page after printing this line;

01-99 - Skip the specified number of lines after printing this line;

00 - Do not advance paper after printing this line;

Blank - Advance paper one line after printing this line;

ES - Eject to the first print line of the next page and stop printing after printing this line.

It is recommended that paper feeding on the Honeywell 800 be accomplished by means of the post-print skip control wherever possible.

COLUMNS 31-32 are not used.

The remaining columns of a Line Action Descriptor are used to specify three types of information whose effects are illustrated in Figure 9. This figure presents a simplified flow chart of the actions which result from performing the statement:

WRITE (Report Name).

The program performs a series of checks for control breaks starting at level 01 and continuing up to the highest included level. At each level where a control break occurs, the necessary tabulation is performed and a print line at that level is assembled. After a line is assembled, a specified source program procedure may be executed if desired. Finally, the printing of the line may be made contingent upon the truth of a pre-print condition statement.

TABULATION CONDITION OR CONTROL FIELD -- columns 33-48: In describing a detail, total, or final total line, these columns may contain the name of the control field at that level or the name of a definition whose truth represents a control break at that level. These columns are not used on descriptors for type TT, NH, or FL lines. In the case of a folded line, they may only be used on the subline 1 descriptor.

A control break is said to occur at a given level if any of the following occurs at that level:

(1) The control field designated at that level changes value (does not apply at detail level);

(2) The tabulation condition named in columns 33-48 is true (a blank tabulation condition is defined as true); or

(3) A control break occurs at a higher level.

PRE-PRINT PROCEDURE -- columns 49-64: The programmer may designate here the name of a source program procedure to be performed prior to printing a line. The normal rules governing procedure names (see page 20) apply. Continuation descriptors may be used if the name exceeds 16 characters. A pre-print procedure is performed after all crossfooting and tabulation have been performed on the current line and immediately before testing the Print Condition (see below). In the case of a folded line, only the descriptor for subline 1 may designate a pre-print procedure.

PRINT CONDITION -- columns 65-80: The programmer may designate here the name of a definition to be tested immediately before printing the current line. Continuation descriptors may be used if the name exceeds 16 characters. When a print condition is tested, its truth allows the line to be printed. In the case of a folded line, a separate print condition may be specified for each included subline.

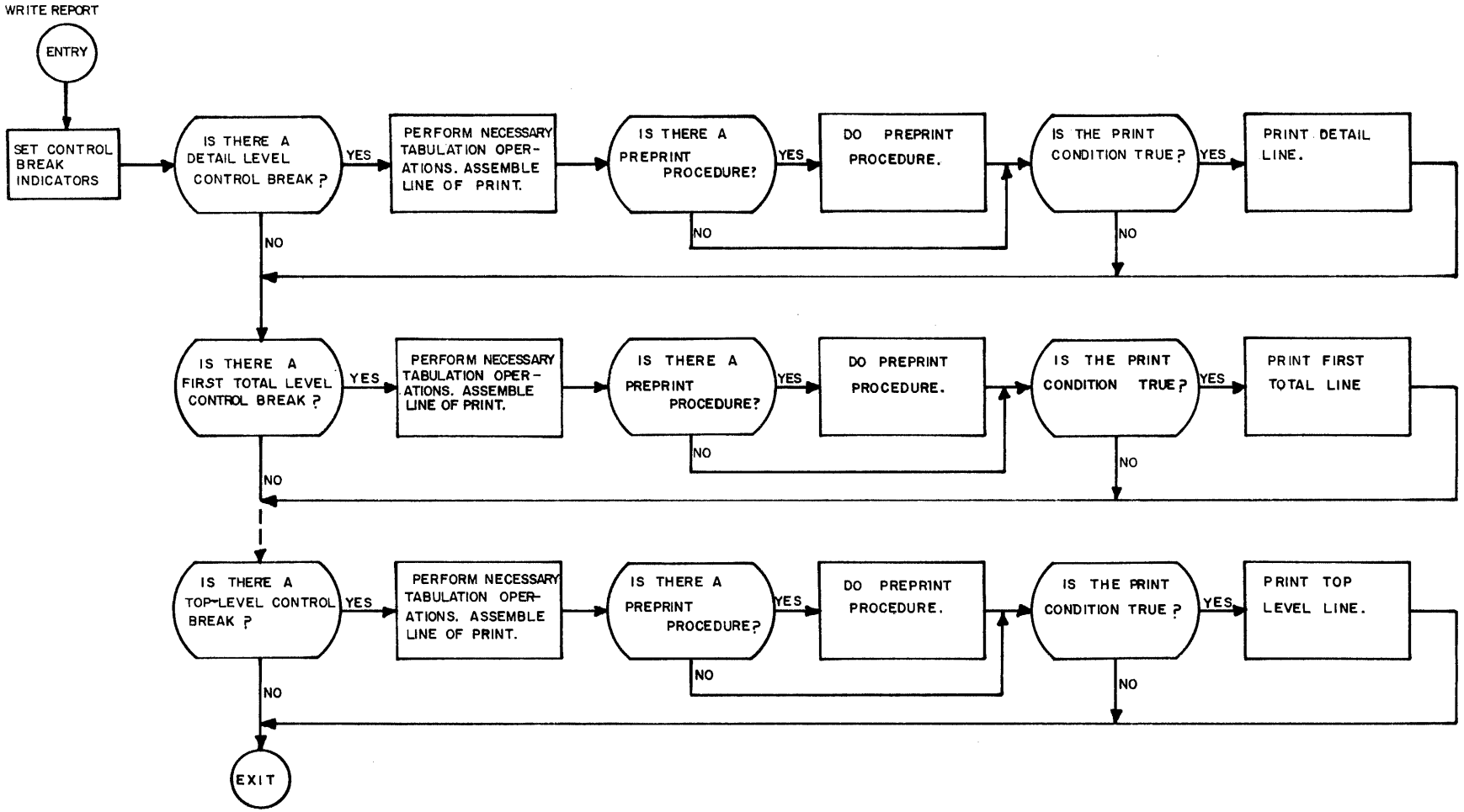


Figure 9. Flow Diagram of Actions Resulting from WRITE Statement

Report Field Descriptors

Each field which is to be used in a report must be described on a Report Field Descriptor, punched from information presented on the Report Description Form (Figure 8).

DESCRIPTOR TYPE -- column 1: Report Field Descriptors are identified by the presence of an "F" punched in this column.

SERIAL NUMBER -- columns 2-6: The use of serial numbers on Report Field Descriptors complies with the standard use of such numbers. If these numbers are assigned, each descriptor presented on the Report Descriptor Form must be serialized according to one over-all numbering sequence (i.e., each Report Name Descriptor followed by the related Line Action Descriptors and each of the latter followed by the related Report Field Descriptors).

FIELD NAME -- columns 7-24: The name of each field to be used in preparing a report is placed in these columns. If the field name exceeds 18 characters, continuation cards may be used. The standard rules for field names apply. If no line layout is prepared, a literal which does not lie within a field is made the contents of an un-named field.

TABULATION ACTION CODE -- column 25: This column is used to specify which fields in a line are to be accumulated and tabulated on control breaks. Permissible codes are T, A, and blank.

T (Tabulate) or A (Accumulate). Either of these action codes directs that the current field is to be tabulated. If one or more fields in a line have either of these action codes, the Line Action Descriptor for that line must designate a control field or a tab condition. Each time that a control break occurs, the accumulated value of all tabulated fields at that or any lower level are used to generate a total line and then cleared to zero. If the accumulated value of a tabulated field exceeds the assigned length of that field, the overflow information is not lost unless the accumulated value exceeds either eleven digits or the allotted number of print positions in the total line, whichever is smaller. A tabulated field may be referenced directly in the source statements by using the name of the line in which it occurs as a modifier for the field. For example, the total grosspay of a department could be referenced either as DEPARTMENT GROSSPAY or as GROSSPAY OF DEPARTMENT.

Blank (do not tabulate). Fields which are described by a blank in column 25 are not tabulated.

FIELD SUPPRESSION -- column 26: Fields which are described by an "S" in column 26 are unconditionally suppressed and must not be represented by a carat in the line layout description. Unconditional field suppression may be used, for example, to suppress quantities used in a crossfooting operation (see columns 27-29 below). Fields which are described by a blank in column 26 are printed normally unless they are conditionally suppressed according to the code in column 30 (below).

CROSSFOOTING -- columns 27-29: Crossfooting is the performance of addition or subtraction on two or more decimal fields which appear in a single report line, the result being inserted into the same report line where it may be referenced by source statements. A crossfoot operation is specified by placing any Hollerith character in the addition column (27) of each field to be summed and the subtraction column (28) of each field to be subtracted and the same character in the result column (29) of the field in which the sum is to appear. Several different crossfoot operations may be performed within a given report line, using a different character to represent each separate operation. If several different crossfoot operations involve the same field, that field may have more descriptors as necessary. Operand and/or result fields may be unconditionally suppressed if desired. A single crossfoot operation may extend over all of the physical print lines comprising a "folded" report line.

Figure 10 illustrates the use of the Field Suppression and Crossfoot columns to set up two multiple crossfoot operations within a single report line, suppress certain operands, and print certain results. The operations designated to be performed in this illustration are:

(1) A - C + D = G

(2) B + E = H

All of the operands in these operations are unconditionally suppressed. Only the results G and H are printed.

Columns 7-24 Field Name	26 Supp.	27 CF+	28 CF-	29 CF=
A	S	X		
B	S	Y		
C	S		X	
D	S	X		
E	S	Y		
F	S			
G				X
H				Y

Figure 10

ZERO AND GROUP SUPPRESSION -- column 30: This column may be used to designate two different types of conditional suppression. Permissible codes are Z, G, and blank.

Z (Zero Suppression). This code applies only to decimal fields. If it is applied to a non-decimal field, it will have no effect on the printing of the field. If the value of a zero-suppressed decimal field is zero, the entire field is replaced by blanks, including literal commas, decimal points, and floated characters, if any.

Blank (Normal Suppression). A blank in this column has no effect on non-decimal fields. With decimal fields, it has the effect of suppressing only zeros to the left of the first non-zero digit. If the value of the field is zero, only a single zero digit is printed in the print position indicated by the carat. In this case, literal commas, decimal points, and floated characters are not printed.

G (Group Suppression). This code applies only to fields which appear in detail lines. If it is applied to a field which does not appear in a detail line, it will have no effect on the printing of the field. A field which is group suppressed is printed only in the first detail line following a total line and is then suppressed until the appearance of the next total line. Group suppression is usually used to suppress identical values of control fields, printing only each new value of the control field in the line in which it first occurs.

SIGN SPECIFICATION -- column 31: This column is used to specify the convention which will represent positive and negative signs in the related field. The following is a list of the permissible codes and the actions which result if the related field contains a negative or a positive sign, respectively:

(1) P or Blank: The position immediately to the right of a signed field will contain either - or blank.

(2) C: The above position will contain either CR or blank.

(3) B: The above position will contain either - or +.

(4) R: The above position will contain either CR or +.

(5) F: The position immediately to the left of the high-order digit in a signed field will contain either - or +.

(6) M: The position immediately to the left of the high-order digit in a signed field will contain either - or blank.

(7) A: Absolute value; no sign information is printed.

(8) H: The card column containing the high-order digit of a signed field will also contain an 11 or 12 overpunch.

(9) L: The card column containing the low-order digit of a signed field will also contain an 11 or 12 overpunch.

The sign codes H and L are provided for specifying sign conventions in punch card reports. If these codes are used in a printed report, the character with the sign will be non-numeric. All other codes may be used in connection with either printed or punched reports.

CHECK DECIMAL -- column 32: If the code "D" appears in this column, a check digit is automatically computed and appended to the right of a decimal field. The check digit prints in the position designated by the carat. If any other code (or no code) appears in this column or if the related field is non-decimal, this column is ignored. The value of the check digit is computed by multiplying the low-order digit of the field by one, and successively higher-order digits by 2, 2^2, etc. These products are summed and 1 is added to the sum. The units digit of the result is the check digit.

INCREMENT -- columns 33-35: These columns may contain a right-justified numeric increment from 000 to 999 which will be added to the contents of the related field before printing. Incrementing may be used for counting, in which case the initial value of the field to be counted is automatically set to zero unless otherwise specified.

SHIFTING -- columns 36-38: These columns may be used to indicate the scaling of decimal fields which are not internally scaled. If the related field is scaled both internally and by an amount stated in these columns, the scaling stated in columns 36-38 is in addition to the internal scaling. The related field is shifted in the manner indicated by the code in column 36 and by the amount specified (right-justified) in columns 37-38. The permissible codes for column 36 are R, T, L, and blank. The effects of these codes are as follows:

(1) R: Shift right the number of places specified in columns 37-38 and round the result;

(2) T: Shift right the number of places specified in columns 37-38 and truncate the result;

(3) L: Shift left the number of places specified in columns 37-38 and fill in the same number of low-order digit positions with zeros;

(4) Blank: Do not shift the related field, except as required for internal scaling.

If columns 37-38 are blank, then column 36 is ignored. If columns 37-38 contain a number and column 36 is blank, the field is not shifted but is rounded at the specified position from the right. All digits to the right of a rounded digit are set to zero. Several examples of shift codes and their results are shown in Figure 11.

Scaled Number	Shift Code	Shifted Number	Field Image	Printed Number
123.456	R01	12.3456	. ^	12.346
1234.56	L02	123456.	. ^	123456.000
1234.56	T02	12.3456	. ^	12.34

Figure 11.

FLOATED CHARACTERS -- column 39: Any character (usually $ or *) placed in this column is printed immediately to the left of the high-order digit in the related field. Floating characters apply only to decimal fields.

MAXIMUM NUMBER OF CHARACTERS -- columns 40-42: If the maximum length of a variable-length field to be printed is different from the maximum length given in the File Outline Form, then the maximum length to be printed is specified in these columns, justified to the right.

LEFT MARGIN -- columns 43-45: The starting print position of a variable-length field is written in these columns, justified to the right. (The ending position may be specified in the Right Margin columns or by means of a carat on the line layout). Variable-length fields are printed left justified at the stated starting position and continue between the designated left and right margins through as many lines as necessary. These columns are not used to describe a fixed-length field.

If the programmer chooses to prepare complete Line Layout Descriptors, then the related Field Descriptors are completed with the preparation of columns 1-45 and columns 46-80 may be left blank. If Line Layout Descriptors are not prepared, columns 46-80 are used to present the layout information.

RIGHT MARGIN -- columns 46-48: The ending print position of a field is written in these columns, justified to the right.

FIELD IMAGE -- columns 49-80: An image of the field is written in these columns, exactly as would have been done on Line Layout Descriptors. The field image is

presented with a carat in the low-order character position and may include literal characters, background, a decimal point, commas separating thousands and millions, etc., as described under Line Layout Descriptors on page 43. The position in the field image which contains the carat corresponds to the ending position specified in the Right Margin columns. If the least-significant character of a literal is to be printed in the right-most position of a field, the image of that field must be justified to the right (i.e., the least-significant character must appear in column 80). If a literal is not to be printed within a field, it must be made an un-named field unless a Line Layout is used.

Execution of Generated Reports

FACT includes a select and print routine which selects and prints the reports generated onto the report tape. The reports on this tape may be printed in any priority which the programmer may specify.

SECTION V

INPUT DATA DESCRIPTION

If the functions of the program to be compiled include input data editing, then the programmer must describe the format of the input data to FACT. Data which is to be used as input to an object program may be punched on standard 80-column cards in an unrestricted number of different formats. Each of these card formats must be completely described on the Card Descriptor Form shown in Figure 12. The description of a data card includes one (or more) lines to identify the card, followed by one (or more) lines to describe each included field. (Here, as elsewhere, the convention applies that a card which contains input information to the object program is called a "card", whereas one which contains input information to FACT is called a "descriptor".) Thus the input which is required by FACT to create an input editing program includes a deck of Card Descriptors, a deck of File Descriptors, and the necessary configuration and control statements.

The Card Descriptor Form consists of the following fields, some of which pertain to every line of the form and some of which pertain only to certain lines.

DESCRIPTOR TYPE -- column 1: A Card Descriptor is identified by the presence of a C punched in column 1.

SERIAL NUMBER -- columns 2-6: The use of serial numbers on the Card Descriptor Form is identical to the use of such numbers on the File Outline Form.

FIELD NAME OR IDENTIFICATION CODE -- columns 7-22: These columns may contain either the name of a field being described or the code which is used to identify a particular card. The use of these columns in any given line is indicated by the contents of the Mode columns (33-34). If the Mode columns contain the punches "ID", then the contents of columns 7-22 represent the punch configurations which identify a specific card, and columns 23-26 define the area of the card in which these identifications appear. For example, a particular data card might be identified by the configuration "276" in columns 1-3. The identification Card Descriptor for this format would then contain the punches "ID" in columns 33-34, "276" in columns 7-9, and "0103" in columns 23-26. More complicated card identifications can be handled by the use of Y and X overpunches in column 26 to represent, respectively, the con-

Honeywell

Electronic Data Processing

CARD DESCRIPTOR FORM

Title . . . . . Revision . . . . .

Prepared By . . . . . For Program . . . . .

Date . . . . . Checked By . . . . .

Remarks . . . . .

FACT

	C	SERIAL NUMBER	FIELD NAME	SIZE FROM	SIZE TO	SELECTOR KEY	SELECTOR FROM	SELECTOR TO	MODE	MODE ERROR	BLANKS	SIGN PLUS	SIGN MINUS	SIGN LOCATION	DECIMAL POINT	NORMAL ALLOTTED LENGTH	MAXIMUM LENGTH	JUSTIFICATION	ACTION CODE	REPORT OR PROCEDURE NAME	ACCEPTANCE CONDITION
1	C																				
2	C																				
3	C																				
4	C																				
5	C																				
6	C																				
7	C																				
8	C																				
9	C																				
10	C																				
11	C																				
12	C																				
13	C																				
14	C																				
15	C																				
16	C																				
17	C																				
18	C																				
19	C																				
20	C																				

FORM NO. T1203

Figure 12

nectives AND and OR. For example, a card might be identified by either of two identification codes: (1) "276" in columns 1-3 and "Q" in column 79; or (2) "138" in columns 1-3 and "P" in column 80. The description of this identification convention requires four descriptors, of which the first is a base descriptor and the others are continuation descriptors. The relevant contents of these four descriptors would then be as follows:

Column	7	8	9	23	24	25	26	33	34
	2	7	6	0	1	0	+3	I	D
	Q			7	9	7	-9	I	D
	1	3	8	0	1	0	+3	I	D
	P			8	0	8	0	I	D

In combining multiple codes of this type, all AND operations are performed before OR operations. In the above example, the existence of a Y or an X overpunch in column 26 is represented, respectively, by the signs + and -.

The identification Card Descriptor for a given card is followed by the series of field Card Descriptors which describe all of the fields comprising that card. A field Card Descriptor is distinguished from an identification Card Descriptor by the existence of any punches other than "ID" in the Mode columns (33-34). Thus the structure of a Card Descriptor deck consists of an identification descriptor followed by all of the descriptors for the related fields; then the identification descriptor of another input card, followed by all of the descriptors of the fields in that card, etc. It is possible that the input data may include one or more fields which are common to all of the input formats involved. The descriptors for such fields may be grouped together following an identification descriptor which contains the punches "ALL" in columns 24-26 and no punches in columns 7-22.

A field Card Descriptor contains the name of the field being described in columns 7-22. Every field must be assigned a name, the length of which is virtually unrestricted. The name, which may include any alphanumeric character (A-Z and 0-9) plus the hyphen, may be punched anywhere in this area as leading and trailing blanks are disregarded. Embedded blanks, however, are not allowed, which means that multiword names must be connected by hyphens, vis., PART-NUMBER, BASIC-PAY-RATE, etc. However, a field name may be modified by adjectives or by prepositional phrases exactly as in source language (Section III). For example, if the File Outline Form includes a field called NUMBER within a group called PART, the input field might be referenced either as PART NUMBER or as NUMBER OF PART. If the

length of the name exceeds 16 characters (including hyphens), continuation descriptors may be used to punch this information.

FIELD LOCATION -- columns 23-26: The starting and ending locations of the field on the input card are punched in columns 23-24 and 25-26 respectively, using 2-digit numbers from 01 to 80. The same number is punched as both start and end location to describe a single-column field. Continuation descriptors may be used to combine several non-continuous columns or groups of columns to form a single field in the tape format. The field name appears only on the first descriptor of such a set, but may be continued if necessary. The first non-continuation descriptor indicates the termination of the set. Fields formed by combining non-contiguous sub-fields are constructed by starting with the column(s) indicated on the base descriptor and appending sub-fields to the right in sequential order.

ROW SELECTOR -- columns 27-32: One or more columns of an input card may be used for more than one purpose. For example, throughout all or any portion of a card, the numeric rows (0-9) may be used to store data while the zone rows (11 and 12) are used for control purposes. In such a case, the Row Selector is used to describe those punches which are to be used (or selected) as part of a data field and the columnar limits within which such row selection will be used. Column 27 may contain one of the punches S, U, or L, while column 28 may contain any single punch. An "S" means that the row specified in column 28 is selected. A "U" means that all the upper rows are selected, down to and including that specified in column 28. An "L" means that all the lower rows are selected, up to and including that specified in column 28. Columns 29-30 and 31-32 specify the columnar limits within which the selection described is effective. Any combination of non-continuous rows may be selected by means of continuation descriptors. The limits specified in columns 29-32 must not lie outside of the field limits specified in descriptor columns 23-26, even though the Row Selection process may be used in more than one input field.

FIELD MODE -- columns 33-34: On an identification Card Descriptor, columns 33-34 contain the code ID. On a field Card Descriptor, any of the following nine modes of input data may be specified in these columns by writing the indicated codes:

Code	Mode	Legal Characters
SP	Single Punch	0-9, 11, or 12
NH	Numeric Hollerith[1]	0-9
A	Alphabetic[1]	A-Z
AN	Alphanumeric[1]	A-Z and 0-9
AS	Alphabetic, Numeric and Sign[1]	A-Z, 0-9, 11, and 12 (or - and +)
H	Hollerith	Any legitimate punch
D	Decimal[2]	0-9 and zone punches for signs
CD	Decimal with Check Digit[3]	0-9
HD	Hexadecimal[4]	0-9 and B-G
OC	Octal[4]	0-7

Single character codes may be punched in either column 33 or 34. For example, the codes "A blank" and "blank A" are equivalent.

For each of the available modes, the Blank Column Convention in column 36 is used to indicate whether or not blanks are acceptable for the field being described.

MODE ERROR -- column 35: Each time that an input field is read from an input card, its mode is compared with the mode specified in descriptor columns 33-34. If this comparison fails, the resulting action is determined by the contents of column 35. Acceptable punches in this column are E, B, Z, S, L, and blank.

E or blank — If column 35 contains an E or a blank, any discrepancy in mode results in an error indication.

Z — If column 35 contains a Z, any incorrect punch is replaced by a zero. In the CD mode, the check digit is corrected if necessary.

B — If this code is specified, any incorrect punch is replaced by a zero in the D, CD, HD, and OC modes. In the NH, SP, A, AN, AS, and H modes, however, an incorrect punch is replaced by a blank. In the CD mode, the check digit is corrected if necessary.

---

1. These modes may be intermixed to form a single field from non-contiguous columns.
2. A decimal field in the Honeywell 800 may contain up to eleven digits.
3. A check decimal field or an unsigned decimal field in the Honeywell 800 may contain up to twelve digits, including the check digit.
4. A hexadecimal or octal field may be up to one machine word in length.

S — This code may be specified only for the D, CD, and OC modes. The smallest legitimate punch in an error column is selected. If no legitimate punch is present, a zero is substituted. In the CD mode, an incorrect check digit after selection results in an error indication.

L — This code may also be specified only for the D, CD, and OC modes. The largest legitimate punch in the error column is selected. If no legitimate punch is present, a zero is substituted. In the CD mode, an incorrect check digit after selection results in an error indication.

BLANK COLUMN CONVENTION -- column 36: This column is used to specify the manner of handling blanks in the input data. Acceptable punches in this column are E, B, Z, I, and blank.

E — If column 36 contains an E, any blank in an input field, regardless of mode, is considered an error.

B or blank — If column 36 contains a B or a blank, any blank in an input field is accepted. If the field is D, CD, HD, or OC, the blank is replaced by a zero; otherwise, it remains a blank.

Z — If this code is specified, any blank in an input field, regardless of mode, is accepted and replaced by a zero.

I — This code may only be specified for the D, HD, OC, or CD mode. Leading and trailing blanks in an input field are accepted and replaced by zeros; embedded blanks are treated as errors.

SIGN (for decimal fields only) -- columns 37-40: These columns may be used to describe a convention for designating the sign of a numeric field. The sign itself may be punched anywhere on the data card. The punch convention specifying plus is punched in column 37; the convention specifying minus is punched in column 38. Columns 39-40 designate the input card column (from 01 to 80) in which the sign conventions are punched. An unsigned decimal field is specified by the code UN in columns 37-38 and blanks in columns 39-40. If the programmer wishes to specify only a plus or a minus convention but not both, column 37 or 38 may be left blank.

If plus and minus are not punched in the same input card column, continuation descriptors may be used. Continuation descriptors may also be used to specify more than two sign conventions for the same field. In the latter case, the related data field may contain any of the specified conventions.

If a numeric field is to be right justified (see below), signs may be positioned immediately to the left of the high-order digit or overpunched in the same input card column as the high-order digit. Similarly, if a numeric field is to be left justified, signs may be positioned immediately to the right of the low-order digit or overpunched in the same column as the low-order digit. The following codes may be punched in columns 39-40 to specify one of these "floating" sign locations:

CODE	INTERPRETATION
LS	Leading sign (preceding high-order digit)
TS	Trailing sign (following low-order digit)
LZ	Leading sign, zone overpunch (on high-order digit)
TZ	Trailing sign, zone overpunch (on low-order digit)

Overpunch sign conventions are restricted to the 12 and 11 punches. The same restriction applies whenever the signs are located anywhere within the numeric field.

If no sign convention is described, the following standard sign convention is assumed:

(1) A field containing no zone punch (11 or 12) is considered positive;

(2) A field containing a 12 punch in either its first or last column is considered positive;

(3) A field containing an 11 punch in either its first or last column is considered negative.

Under this convention, the column containing a sign overpunch may also contain a numerical punch representing one digit of the field. Obviously, a blank may be used as a positive but not as a negative sign convention.

DECIMAL POINT -- columns 41-42: These columns are used only with decimal fields to indicate the position of the decimal point. If these columns are blank, the decimal point is assumed to be immediately to the right of the low-order digit. If these columns are used, the position of the decimal point is indicated relative to the above position. A 2-digit number is written, right justified, to indicate the amount by which the decimal point is removed from the above position. An 11 overpunch in column 42 or a scale number without an overpunch indicates that the decimal point is to the left of the low-order digit. A 12 overpunch in column 42 indicates that the decimal point is further to the right of the low-order digit.

NORMAL ALLOTTED LENGTH -- columns 43-45: In describing a variable-length field, these columns specify a number of characters sufficient to accommodate most appearances of the field. In describing a fixed-length field, these columns are not used.

MAXIMUM LENGTH -- columns 46-48: The maximum length of a variable-length input field must be presented in these columns, justified to the right, just as is done on the File Outline Form for fields which are created internally. Therefore, if columns 46-48 are used, columns 43-45 must also be used and the presence of punches in these two areas defines a field as variable-length.

JUSTIFICATION (for numeric fields only) -- column 49: This column is used to describe the justification of an input field. Acceptable punches are L, R, and blank.

L — L punched in these columns specifies that the input field is left justified; i.e., the high-order digit is punched in the left-most column of the field. Blanks to the left of a left-justified field are treated as errors, regardless of the Blank Column Convention specified. Blanks to the right of a left-justified field are replaced by zeros.

R — An R punched in these columns specifies that the input field is right-justified; i.e., the low-order digit is punched in the right-most column of the field. Blanks to the right of a right-justified field are treated as errors, regardless of the Blank Column Convention specified. Blanks to the left of a right-justified field are replaced by zeros.

Blank — A blank in these columns specifies that the input field is not justified and that the treatment of all blanks is governed by the Blank Column Convention specified.

ACTION CODE -- column 50: The manner in which information from data cards is filed on tape is determined by the structure described on the File Outline Form, the action codes assigned, and the order in which the data is received. The legitimate codes which may appear in column 50 are C, E, R, and blank, which are defined as follows:

C — Any field which has an action code of C is called a "change control field". This means that a change in the value of this field from the preceding value constitutes a control break and causes information to be filed.

E — Any field which has an action code of E is called an "existence control field". This means that each occurrence of this field constitutes a control break and causes information to be filed.

R — Any field which has an action code of R is called a "replacement

field". This means that the current value of this field replaces the preceding value internally. Such a field never constitutes a control break and never causes information to be filed.

Blank — If column 50 is blank, FACT assigns an action code to the corresponding field as follows: (1) if the field is part of a secondary group which contains no secondary subgroups (as shown on the File Outline Form), an action code of "E" is assigned; (2) otherwise, an action code of "C" is assigned.

The action codes assigned by FACT in the absence of programmer-specified codes are designed to handle the great majority of applications. Thus the programmer is not normally required to use column 50.

When a card-editing program is executed, the file structure specified by the File Outline Form is established within the computer. As cards are read, the information obtained from them is inserted into the proper locations in this structure by relating the names of the input fields to the corresponding names in the file outline. Thus the first occurrence of each field merely serves to fill out the file skeleton internally and transfers no information to the file tape. As each field is read from an input card, the program examines the skeleton to see if this field is already represented by previous information. If not, then the value just read is inserted at the proper point in the skeleton. However, if the corresponding field of the skeleton contains a previously read value, then the program tests the action code and the new value of the field to see if a control break has occurred. (Note that a control break may be any occurrence of an E-type field or a change in the value of a C-type field; therefore, the absence of a control break may be any occurrence of an R-type field or a repetition of the previous value of a C-type field.) If a control break has not occurred, the field value just read is inserted into the skeleton, replacing the previously stored value, and no information is filed on tape. If a control break has occurred, all information stored in the skeleton is filed on tape, except that which has been previously filed. However, not all of the skeleton is cleared to receive new information. That portion of the skeleton which is cleared consists of the smallest secondary group containing the field which caused the control break. This group is entirely cleared, including all of its subordinate groups and fields. Note that the related secondary group is the governing factor in file creation. In this connection, the difference between primary and secondary groups is critical because it determines the effects which subsequently read fields will have on file creation. After clearing, the field which caused the con-

trol break is stored in the skeleton. The technique of filing data is diagrammed in Figure 13.

For example, Figure 14 shows the structure of a tape file to be written and the formats of three types of input cards to be read, all in very simplified form. In this figure, an entry (A) is composed of a number of primary groups followed by a secondary group (H). This group, in turn, is composed of two fields (I and J) and a secondary subgroup (K) which contains the two fields (L and M). The file is to be built up by reading and editing cards of the types X, Y, and Z, whose contents are also listed in Figure 14. If no action codes are provided, the Compiler assigns type E to fields L and M, since they are the only fields which are part of a secondary group containing no secondary subgroups, and type C to all of the other input fields. Consider the sequence of file creation if action codes are so assigned.

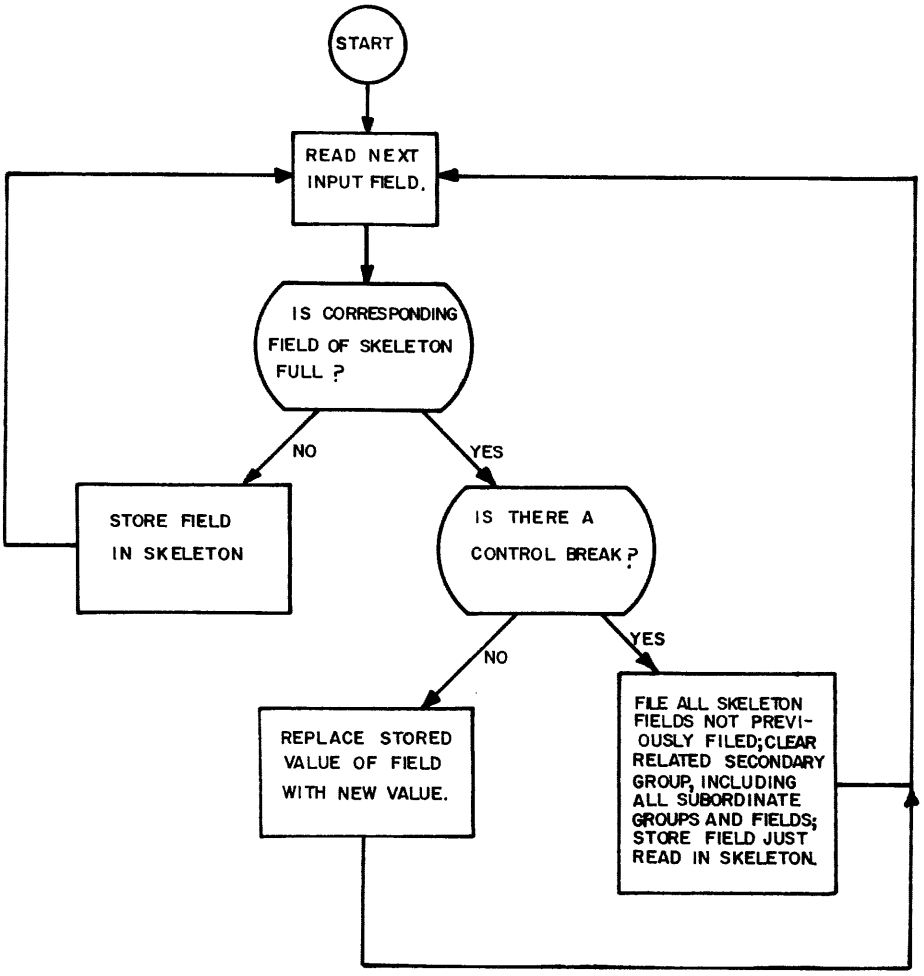


Figure 13

File Outline		Input Card Type	Contents
A		X	B D F G
B	(type C)		
C		Y	B I J
D	(type C)		
E		Z	B I L M
F	(type C)		
G	(type C)		
*H			
I	(type C)		
J	(type C)		
*K			
L	(type E)		
M	(type E)		

Figure 14

The normal order which can be anticipated in the input deck is an X card, followed by a number of Y and Z cards, each Y card being followed by a number of Z cards. This is just another way of saying that lower-level information normally changes more often than higher-level information. Assuming this general arrangement of the input deck, then, the first three cards read will probably be an X, a Y, and a Z in that order. The information derived from these three cards will just fill out the internal skeleton. The next card is probably a Z card which causes the filing of the entire skeleton, the clearing of secondary group (K), and the refilling of this group with new information. Another Z card then causes filing only of the new group (K), since the balance of the entry is already filed, and again clears and refills this group. When a Y card is encountered, a control break can be assumed, since a card containing only previously read information is unlikely. This, then, causes the filing of all information not previously filed (in this case, the last K group), the clearing of the entire group (H), and the refilling of fields (I) and (J). When an X card is encountered, a control break may be assumed for the same reason, and thus such a card will cause all unfiled information to be filed and the entire entry cleared for new information. The reader should note that the foregoing brief discussion of the example does not cover every case which may arise, since every sequence of X, Y, and Z cards is actually a special case. However, the actions which result from any such sequence can be determined by applying the stated rules for file creation.

REPORT OR PROCEDURE NAME -- columns 51-65: During the execution of a card-editing program, the editing of any input field may be immediately followed by the performance of any procedure which is defined on the Program Statements Form. This technique, which is specified by writing the name of the desired procedure in columns 51-65 of the Card Descriptor, allows the programmer to integrate program statements with card-editing operations to achieve considerable editing sophistication.

A specific instance of the use of these columns is batch control, which is a technique commonly used to check the accuracy of input data. Batch control is achieved by integrating the card-editing function with the report writer and the program statements in the following manner. Detail data cards are read in batches, each batch being followed by a uniquely identified total card which contains the batch totals of all fields which are to be checked. A report is described on the Report Description Form which consists of the input fields to be batch totalled. The tabulation controls described in Section IV are used to total these fields. This report is normally a dummy report; that is, the batch totals are never printed but are merely used for comparison with the totals read from the total card. Each input field is described on the Card Descriptor Form in the usual manner, except that the description of the rightmost field to be checked contains the name of the dummy report in columns 51-65. Thus, as the detail cards are read, the report writer is set up to accumulate the specified fields.

A source procedure must be prepared to check the batch totals against the total cards. Appropriate action must be specified for balanced and out-of-balance batches. Normally a balanced batch is accepted and editing proceeds with the next batch. A typical corrective action for an out-of-balance batch might consist of reversing the file tape, transferring the bad batch to an error-file tape, reversing the file tape again, and continuing with the following batch. The procedure which does the comparing and takes appropriate action is named in columns 51-65 of the batch-total descriptor. If a single balance procedure is to be used to check all of the fields on a batch total card, the procedure is named on the last batch-total descriptor. Alternatively, separate procedures may be used to balance different fields. In summary, each time that the object program identifies and reads a detail card, it enters the report writer to accumulate the specified fields. Each time that the object program identifies and reads a total card, it enters the specified procedure to perform a batch check and any necessary resulting action.

ACCEPTANCE CONDITION -- columns 66-80: These columns may contain the name of a definition whose truth determines the acceptance of the input field. This technique may be used to achieve many types of input validity checking in addition to the normal checks for field length and mode. The acceptance statement might specify a series of legitimate values for the input field, a range of legitimate values, or a series of such ranges. For example, a field called RATE may contain any of the following legitimate values: 175, 200, 225, 250, or 275. The descriptor for this field contains the name CHECKRATE in columns 66-80. Each time that this field is read from an input card, the following definition is referenced.

CHECKRATE. RATE IS 175, OR 200 OR 225 OR 250 OR 275.

If this definition is correct, the field is accepted; otherwise, it is flagged as an error. If an input card format contains several detail fields plus a crossfoot total, this total may be checked during editing by means of an acceptance condition.

Overflow Card Fields

It may be necessary to describe on the Card Description Form a field which is too long to be contained on a single card. This may be a fixed-length field which is divided between two or more input cards, or it may be a variable-length field which in its longer appearances is divided between two or more input cards. In either of these cases, it is necessary to prepare a special descriptor, called a subsidiary identification descriptor, to describe the overflow field.

A subsidiary identification descriptor contains the same information as any other identification descriptor plus a continuation overpunch in column 6. It must immediately follow the card descriptor (or the last continuation descriptor) pertaining to the overflow field. The subsidiary ID descriptor (or its last continuation) must be immediately followed, in turn, by an overflow card descriptor which specifies the columns occupied by the field on the overflow card, using the Field Location columns (23-26). The Field Name columns on this descriptor are left blank.

If the overflow field is variable in length, this pair of continuation cards is sufficient to describe any number of overflow cards, limited only by the maximum field length stated on the base descriptor. If the overflow field is fixed in length, a similar pair of continuation cards (subsidiary ID and overflow descriptor) is required for each overflow card comprising the entire field. All of these continuation cards must contain the customary overpunch in column 6. If serial numbers are assigned, subsidiary ID and overflow descriptors are simply numbered in the normal sequence. The cor-

responding input cards which contain the various portions of an overflow field must be presented to the computer in the correct order.

Name Association

As has already been described, an input editing run may reference the FACT report writer, by means of the Report or Procedure columns, to perform certain operations on the input data. Sometimes the report writer is used during input editing to generate information which is to be included in the file being written.

For example, a file of bond redemptions is being written for use in updating a master file of bond records. The information describing three different bonds is punched in each redemption card. The input run which prepares the redemption file includes the necessary batch control provisions to accumulate vertical totals on each of the three columns of bonds being read simultaneously. Each batch of redemption cards is followed by a total card containing three columnar totals to be compared with the totals accumulated internally. To obtain the actual total value of bonds represented by each batch, it is necessary to crossfoot the three vertical totals, which can be accomplished by the report writer. This produces a total which is available internally but which has no name that can be referenced by source language. To make this total available to the source language, it must be added to the redemption file outline. The programmer inserts in this outline a phrase which specifies the batch total, followed by a legitimate field name in parentheses, as follows:

FINAL TOTAL OF BATCH-TOTAL REPORT (TOTAL)

SECTION VI
SAMPLE APPLICATION

A simple payroll application is shown here to demonstrate the use of FACT inputs in problem preparation. The sample payroll is illustrative only and not intended as a general solution of payroll problems.

The sample payroll runs in two phases: one phase prepares a data file from cards; the other updates the master file and calculates the payroll.

The card processing phase creates a data tape from the following types of input cards:

0. Date.
1. New employee records.
2. Changes to be made to the master file.
3. Deletions to be made from the master file.
4. Hours worked by each employee.
5. Batch totals.

The hours cards are batched and each batch is followed by a batch-total card. Each total card contains the batch number, the sum of the HOURS field of the detail cards, and the count of cards in the batch. Batch checks are performed to verify that the detail cards balance with the batch-total card. An error routine removes the bad batches from the output tape and lists them in an ERROR-REPORT.

The second phase of the payroll run begins by sorting the input on employee number. Using the Update function, the detail file is then run against the master file, adding new items, changing existing items, making deletions, and calculating the payroll. Besides the updated master file, output from the maintenance run consists of:

1. Paychecks
2. Bond orders
3. Listings of deletions, unprocessed master-file entries, and erroneous detail items.

The error report is illustrated twice. First, it is worked out completely using the line layout format with fields and lines listed on the Report Description form. Then the same report is completely worked out on the Report Description Form, utilizing the field images.

One page of source statements is presented using the Source Program Statement Form. In addition, the entire source program is shown in the form of a card listing.

Honeywell

Electronic Data Processing

CARD DESCRIPTOR FORM

Title . . . . . Revision . . . . .

Prepared By . . . . . For Program SAMPLE PAYROLL . . . . .

Date . . . . . Checked By . . . . .

Remarks . . . . .

FACT

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80

	C	SERIAL NUMBER	FIELD NAME	SIZE FROM	SIZE TO	SELECTOR KEY	SELECTOR FROM	SELECTOR TO	MODE	MODE ERROR	SIGN BLANKS	SIGN PLUS	SIGN MINUS	SIGN LOCATION	DECIMAL POINT	NORMAL ALLOTTED LENGTH	MAXIMUM LENGTH	JUSTIFICATION ACTION CODE	REPORT OR PROCEDURE NAME	ACCEPTANCE CONDITION
1	C	1	O	7	7				ID											
2	C	2	MONTH	4	9				SP		E									
3	C	3	DAY	10	11				D		E	UN								
4	C	4	YEAR	12	13				D		E	UN								
5	C	5	I	7	7				ID											
6	C	6	EMPLOYEE-NO	1	6				CD		E	UN						C		
7	C	7	NW NAME	8	23				H											
8	C	8	NW RATE	24	28				D			UN			2					
9	C	9	NW EXEMPTIONS	29	30				D			UN			2					
10	C	10	NW BONDEDUCT	31	35				D			UN			2					
11	C	11	NW BOND-DENOMINA																	
12	C	11	TION	36	40				D			UN			2					
13	C	12	2	7	7				ID											
14	C	13	EMPLOYEE-NO	1	6				CD		E	UN						C		
15	C	14	CH NAME	8	23				H											
16	C	15	CH RATE	24	28				D			UN			2					
17	C	16	CH EXEMPTIONS	29	30				D			UN			2					
18	C	17	CH BONDEDUCT	31	35				D			UN			2					
19	C	18	CH BOND-DENOMINA																	
20	C	18	TION	36	40				D			UN			2					

FORM NO. T1203

CARD DESCRIPTOR FORM

Honeywell
Electronic Data Processing

Title . . . . . Revision . . . . .
Prepared By . . . . . For Program SAMPLE PAYROLL . . . . .
Date . . . . . Checked By . . . . .
Remarks . . . . .

FACT

C	SERIAL NUMBER	FIELD NAME	SIZE FROM	SIZE TO	SELECTOR KEY	SELECTOR FROM	SELECTOR TO	MODE	MODE ERROR	SIGN BLANKS	SIGN PLUS	SIGN MINUS	SIGN LOCATION	DECIMAL POINT	NORMAL ALLOTTED LENGTH	MAXIMUM LENGTH	JUSTIFICATION	ACTION CODE	REPORT OR PROCEDURE NAME	ACCEPTANCE CONDITION
1 C	19	CH BONDACCUM	41	45				D			U	N		2						
2 C	20	CH ACCUMGROSS	46	52				D			U	N		2						
3 C	21	CH ACCUMTAX	53	59				D			U	N		2						
4 C	22	CH ACCUMFICA	60	64				D			U	N		2						
5 C	23	CH ACCUMINSUR	65	67				D			U	N		2						
6 C	24	CH ACCUM-RETIREM																		
7 C	24	52 ENT	71	76				D			U	N		2						
8 C	25	3		7	7			I	D											
9 C	26	EMPLOYEE-NO		1	6			C	D	E	U	N						C		
10 C	27	DEL EMPLOYEE		1	6			C	D	E	U	N								
11 C	28	4		7	7			I	D											
12 C	29	EMPLOYEE-NO		1	6			C	D	E	U	N								
13 C	30	RP HOURS		81	1				D		U	N		1					SUMMATION	
14 C	31	5		7	7			I	D											
15 C	32	BATCH-SUM		81	5				D		U	N		1						
16 C	33	BATCH-NUMBER	18	20					H		U	N							BATCH-CHECK	
17 C	34	BATCH-COUNT	21	24					D		U	N								
18 C																				
19 C																				
20 C																				

FORM NO. T1203

FILE OUTLINE FORM

Honeywell
Electronic Data Processing

Title . . . . . Revision . . . . .
Prepared By . . . . . For Program SAMPLE . PAYROLL . . . . .
Date . . . . . Checked By . . . . .
Remarks . . . . .

FACT

	0	SERIAL NUMBER	NORMAL ALLOTTED LENGTH	MAX. LENGTH OR NUMBER OF APPEARANCES	MODE	DECIMAL POINT	ROUND	FILE ENTRY	NAME (DECREASING RANK →)
1	0	1			RU			DETAIL-FILE, (DETAIL), (D)	
2	0	2						DATE	
3	0	3						MONTH	
4	0	4						DAY	
5	0	5						YEAR	
6	0	6						*EMPLOYEE-RECORD	
7	0	7						EMPLOYEE-NO, (EMPLOYNO)	
8	0	8						*NEW-EMPLOYEE, (NW)	
9	0	9						NAME	
10	0	10						RATE	
11	0	11						EXEMPTIONS, (EXEMPT)	
12	0	12						BOND-DEDUCT, (BONDEDUCT)	
13	0	13						BOND-DENOMINATION, (BONDENOM)	
14	0	14						*CHANGE, (CH)	
15	0	15						NAME	
16	0	16						RATE	
17	0	17						EXEMPTIONS, (EXEMPT)	
18	0	18						BOND-DEDUCT, (BONDEDUCT)	
19	0	19						BOND-DENOMINATION, (BONDENOM)	
20	0	20						BOND-ACCUMULATION, (BONDACCUM)	

FORM NO. T1204

FILE OUTLINE FORM

Honeywell
Electronic Data Processing

Title . . . . . Revision . . . . .
Prepared By . . . . . For Program SAMPLE PAYROLL
Date . . . . . Checked By . . . . .
Remarks . . . . .

FACT

	0	SERIAL NUMBER	NORMAL ALLOTTED LENGTH	MAX. LENGTH OR NUMBER OF APPEARANCES	MODE	DECIMAL POINT	ROUND	FILE ENTRY	NAME (DECREASING RANK →)
1	0	21							ACCUM-GROSS (ACCGROSS)
2	0	22							ACCUM-TAX (ACCTAX)
3	0	23							ACCUM-FICA (ACCFICA)
4	0	24							ACCUM-INSUR, (ACCINSUR)
5	0	25							ACCUM-RETIREMENT (ACCRET)
6	0	26							*REGULAR-PAY, (RP), (REG-PAY)
7	0	27							HOURS
8	0	28							*DELETION (DEL)
9	0	29							EMPLOYEE-NO (EMPLOYNO)
10	0	30			I			INTERNAL-FILE1	
11	0	31						WORKING-DATA	
12	0	32		11	D			BATCH-SUM	
13	0	33		11	D			BATCH-NUMBER	
14	0	34		11	D			BATCH-COUNT	0
15	0	35		11	D	1		SUM-OF-HOURS	
16	0	36		11	D			CARDS-IN-BATCH	0
17	0	37			I			INTERNAL-FILE2	
18	0	38						WORKING-STORAGE	
19	0	39		7	H			ACTION	..NONE..
20	0	40		16	H			ERROR-TYPE	

FORM NO. T1204

FILE OUTLINE FORM

Honeywell

Electronic Data Processing

Title . . . . . Revision . . . . .

Prepared By . . . . . For Program SAMPLE PAYROLL

Date . . . . . Checked By . . . . .

Remarks . . . . .

FACT

	0	SERIAL NUMBER	NORMAL ALLOTTED LENGTH	MAX. LENGTH OR NUMBER OF APPEARANCES	MODE	DECIMAL POINT	ROUND	FILE ENTRY	NAME (DECREASING RANK →)
1	0	41			D	2	R		TEMP-HOURS
2	0	42			D	2	R		TEMP-GROSS
3	0	43			D	2	R		TEMP-TAX
4	0	44			D	2	R		TEMP-FICA
5	0	45			D	2	R		INSUR-PREM (INSPREM)
6	0	46			D	2	R		RETIRE-PREM (RETPRE)
7	0	47			D				NUM
8	0	48							PRINTLINE
9	0	49		8					ERROR-ITEM, (E)
10	0	50							EMPLOYEE-NO (EN)
11	0	51							REPORT-HOURS (RH)
12	0	52			U				MASTER-FILE (M), NEW-MASTER (N), OLD-MASTER-FILE (O)
13	0	53							DATE
14	0	54							MONTH
15	0	55							DAY
16	0	56							YEAR
17	0	57							*PAYROLL-RECORD (PR)
18	0	58							EMPLOYEE-NO (EMPLOYNO)
19	0	59							NAME
20	0	60							RATE

FORM NO. T1204

FILE OUTLINE FORM

Honeywell
Electronic Data Processing

Title . . . . . Revision . . . . .
Prepared By . . . . . For Program SAMPLE PAYROLL . . . . .
Date . . . . . Checked By . . . . .
Remarks . . . . .

FACT

	0	SERIAL NUMBER	NORMAL ALLOTTED LENGTH	MAX. LENGTH OR NUMBER OF APPEARANCES	MODE	DECIMAL POINT	ROUND	FILE ENTRY	NAME (DECREASING RANK →)
1	0	61							EXEMPTIONS (EXEMPT)
2	0	62							BOND-DEDUCT (BONDEDUCT)
3	0	63							BOND-DENOMINATION (BONDENOM)
4	0	64							BOND-ACCUMULATION (BONDACCUM)
5	0	65							ACCUM-GROSS (ACCGROSS)
6	0	66							ACCUM-TAX (ACCTAX)
7	0	67							ACCUM-FICA (ACCFICA)
8	0	68							ACCUM-INSUR (ACCUM-INSUR) (ACCINSUR)
9	0	69							ACCUM-RETIREMENT (ACCRET)
10	0								
11	0								
12	0								
13	0								
14	0								
15	0								
16	0								
17	0								
18	0								
19	0								
20	0								

FORM NO. T1204

SOURCE STATEMENTS FOR SAMPLE PAYROLL

```
P  1 NOTE.  SAMPLE PROGRAM - A FILE MAINTENANCE PAYROLL PROGRAM DONE IN TWO
P  2       PHASES, A CARD READING PHASE (I) AND A FILE MAINTENANCE - PAYROLL CAL-
P  3       CULATION PHASE (II).
P  4            PHASE II.
P  5
P  6            SORT DETAILFILE.  (THE TAPE FILE PREPARED BY PHASE I IS SORTED ON
P  7       EMPLOYEE NUMBER.)  OPEN DETAILFILE, OLD-MASTERFILE.  PUT DETAIL DATE
P  8       INTO N DATE.  FILE NEW-MASTERFILE.
P  9
P 10 NOTE.  THE UPDATE FUNCTION WILL BE USED TO PROCESS THIS PAYROLL FILE.
P 11
P 12            UPDATE OLD-MASTERFILE BY DETAILFILE, CONTROL ON EMPLOYNO.
P 13
P 14 MATCHED-MASTER PROCEDURE.
P 15            SEE PROCESS PROCEDURE.
P 16
P 17 MATCHED-NEW-MASTER PROCEDURE.
P 18            SEE PROCESS PROCEDURE.
P 19
```

```
P  20 UPDATED-MASTER PROCEDURE.
P  21           IF ACTION IS NOT ..PAID   .. OR ..DELETED.. PUT ..NO CHECK OR DEL
P  22       .. INTO ERROR-TYPE AND WRITE ERROR FROM PR.  PUT ..NONE   .. INTO
P  23       ACTION.
P  24
P  25 UPDATED-NEW-MASTER PROCEDURE.
P  26           SEE UPDATED-MASTER PROCEDURE.
P  27
P  28 NEW-MASTER PROCEDURE.
P  29           IF NEW-EMPLOYEE PUT NEW-EMPLOYEE AND EMPLOYNO INTO PAYROLL-
P  30       -RECORD, AND PUT ZEROS INTO BONDACCUM, ACCGROSS, ACCTAX, ACCINSUR,
P  31       ACCFICA, AND ACCRET OF PAYROLL-RECORD,  OTHERWISE PUT ..NONEXIST. EMP-
P  32       L  .. INTO ERROR-TYPE, WRITE ERROR FROM DETAILFILE, AND DELETE NEW-
P  33       -MASTER ENTRY.
P  34
P  35 UNMATCHED-MASTER PROCEDURE.
P  36           PUT  ..UNMATCHED MASTER.. INTO ERROR-TYPE AND WRITE ERROR FROM
P  37       PR.
P  38
P  39 UNORDERED-MASTER PROCEDURE.
P  40           PUT ..UNORDERED MASTER.. INTO ERROR-TYPE AND WRITE ERROR FROM PR.
P  41
```

```
P  42 UNORDERED-DETAIL PROCEDURE.
P  43           PUT ..UNORDERED DETAIL.. INTO ERROR-TYPE AND WRITE ERROR FROM D.
P  44
P  45 PROCESS PROCEDURE.  IF REGULAR-PAY SEE PAYCALCULATION PROCEDURE AND LEAVE
P  46      PROCEDURE.  IF CHANGE SEE MASTER-CHANGE PROCEDURE AND LEAVE PROCEDURE.
P  47      IF DELETION SEE MASTER-DELETE PROCEDURE AND LEAVE PROCEDURE.  PUT
P  48      ..ILLEGAL TYPE 1  .. INTO ERROR-TYPE AND WRITE ERROR FROM D.
P  49
P  50 MASTER-CHANGE PROCEDURE.  PUT CHANGE EXCEPT BLANK FIELDS INTO PAYROLL-
P  51      -RECORD.  IF D YEAR IS GREATER THAN OLD YEAR PUT ZEROS INTO PR
P  52      ACCGROSS, ACCTAX, AND ACCFICA.
P  53
P  54 MASTER-DELETE PROCEDURE.  WRITE DELETIONS, DELETE PR ENTRY, AND PUT
P  55      ..DELETED.. INTO ACTION.
P  56
P  57 PAYCALCULATION PROCEDURE.  IF ACTION IS ..PAID   .. PUT ..2ND REG-PAY CARD.
P  58      . INTO ERROR-TYPE, WRITE ERROR FROM DETAILFILE AND LEAVE PROCEDURE.
P  59
P  60           OVERTIME.  HOURS IS GREATER THAN 40.
P  61           SET TEMP-GROSS TO RATE * (HOURS + .5 * OVERTIME * (HOURS - 40)).
P  62      IF N YEAR IS GREATER THAN OLD YEAR PUT ZEROS INTO OLD ACCGROSS, OLD
P  63      ACCTAX, AND OLD ACCFICA.  ADD TEMP-GROSS TO ACCGROSS.
```

```
P 64
P 65          PAY-TAX.  TEMP-GROSS IS GREATER THAN 13 * EXEMPT.  SET TEMP-TAX
P 66      TO PAY-TAX * .18 * (TEMP-GROSS - 13 * EXEMPT).  ADD TEMP-TAX TO
P 67      ACCTAX.
P 68
P 69          IF ACCFICA PLUS .025 TIMES TEMP-GROSS IS LESS THAN 120 SET TEMP-
P 70      -FICA TO .025 TIMES TEMP-GROSS, OTHERWISE SET TEMP-FICA TO 120 LESS
P 71      ACCFICA.  ADD TEMP-FICA TO ACCFICA.
P 72
P 73          SET INSPREM TO .75 PLUS (.22 TIMES OLD RATE).  SET RETPRE TO .55
P 74      PLUS (1.15 TIMES RATE).  ADD INSPREM TO ACCINSUR.  ADD RETPRE TO
P 75      ACCRET.
P 76
P 77          ADD BONDEDUCT TO BONDACCUM.  IF BONDACCUM IS LESS THAN BONDENOM
P 78      SET NUM TO ZERO, OTHERWISE SET NUM TO BONDACCUM DIVIDED BY BONDENOM.
P 79      SUBTRACT NUM TIMES BONDENOM FROM BONDACCUM.  DO BOND PROCEDURE UNTIL
P 80      NUM IS ZERO.
P 81
P 82          WRITE CHECK FROM MASTERFILE AND PUT ..PAID   .. INTO ACTION.
P 83 NOTE.  NETPAY IS COMPUTED BY REPORT CROSSFOOTING.
P 84
P 85 BOND PROCEDURE.  WRITE BONDORDER AND SUBTRACT 1 FROM NUM.
P 86
```

```
P  87 NOTE.  PHASE I OF SAMPLE PROGRAM.  THE FOLLOWING PROCEDURES ARE USED TO
P  88       MAKE BATCH CHECKS DURING THE CARD READING PASS.
P  89
P  90 SUMMATION PROCEDURE.  ADD RP HOURS TO SUM-OF-HOURS.  ADD 1 TO CARDS-IN-
P  91       -BATCH.
P  92
P  93 BATCH-CHECK PROCEDURE.  IF BATCH-SUM IS NOT EQUAL TO SUM-OF-HOURS OR BATCH-
P  94       -COUNT IS NOT EQUAL TO CARDS-IN-BATCH SEE BAD-BATCH.  SET SUM-OF-HOURS
P  95       AND CARDS-IN-BATCH TO ZERO.
P  96
P  97 BAD-BATCH PROCEDURE.  REVERSE NEW-MASTER.  CLOSE PAGE OF ERROR-REPORT.
P  98
P  99            L.  PUT ZEROS INTO PRINTLINE.  SET NUM TO 8.
P 100
P 101            BUILD.  PUT EMPLOYNO AND RP HOURS INTO (NUM)TH EN AND EH.  SUB-
P 102       TRACT 1 FROM CARDS-IN-BATCH AND NUM.  IF CARDS-IN-BATCH IS ZERO WRITE
P 103       ERROR-REPORT, REVERSE NEW-MASTER, LEAVE PROCEDURE.  GET NEXT GROUP.
P 104       IF NUM IS ZERO WRITE ERROR-REPORT AND GO TO L, OTHERWISE RETURN TO
P 105       BUILD.  END OF PROCEDURE.
```

Honeywell
Electronic Data Processing

REPORT DESCRIPTION FORM

Title ....
Prepared By ....
Date ....
Remarks ....
Revision ....
For Program SAMPLE PAYROLL.
Checked By ....

FACT

Line	R/A/F	Ser. No.	Lines per Page	Report Name / Line Name / Field Name	Type or Level	Field Supp.	Pre-Print Skip	Post-Print Skip	Increment
1	R	1	40	ERROR-REPORT					
2	A	2		PAGE-HEADING	H	D	EJ	2	
3	F	3		PAGE-NUMBER					1
4	A	4		COLUMN-HEADINGS	H	D		3	
5	A	5		ERROR-LINE	0	0		1	
6	F	6		1ST EN					
7	F	7		1ST RH					
8	F	8		2ND EN					
9	F	9		2ND RH					
10	F	10		3RD EN					
11	F	11		3RD RH					
12	F	12		4TH EN					
13	F	13		4TH RH					
14	F	14		5TH EN					
15	F	15		5TH RH					
16	F	16		6TH EN					
17	F	17		6TH RH					
18	F	18		7TH EN					
19	F	19		7TH RH					
20	F	20		8TH EN					

Other form columns (blank): Cross Foot (+ − =), Z/G, Sign, Check Decimal, Tab Condition or Control Field, Shift (Type, Extent), Float Char., Max. No. of Char., Left Margin, Right Margin, Pre-Print Procedure, Field Image, Print Condition.

FORM NO. T1207

Honeywell

Electronic Data Processing

REPORT DESCRIPTION FORM

Title . . . . Revision . . . .

Prepared By . . . . For Program SAMPLE PAYROLL . . . .

Date . . . . Checked By . . . .

Remarks . . . .

FACT

R / A / F	SER. NO.	LINES PER PAGE / REPORT NAME · LINE NAME · FIELD NAME	TYPE OR LEVEL	PRE-PRINT SKIP	POST-PRINT SKIP	INCREMENT
F	21	8TH RH				
R	101	40 BONDORDER				
A	102	BOND-HEADING	HDEJ		2	
F	103	BPAGE				1
A	104	BONDORDER-LINE	00		1	
F	105	M EMPLOYNO				
F	106	M NAME				
F	107	N MONTH				
F	108	N DAY				
F	109	N YEAR				
F	110	M BONDENOM				
R	111	36 DELETIONS				
A	112	DEL-HEADINGS	HDEJ		2	
F	113	DPAGE				1
A	114	DELETIONS-LINE	00		1	
F	115	M EMPLOYNO				
F	116	N MONTH				
F	117	N DAY				

FORM NO. T1207

REPORT DESCRIPTION FORM

Honeywell
Electronic Data Processing

FACT

Title . . . . Revision . . . .
Prepared By . . . . For Program SAMPLE PAYROLL
Date . . . . Checked By . . . .
Remarks . . . .

	R / A / F	SER. NO.	LINES PER PAGE / REPORT NAME / LINE NAME / FIELD NAME	TYPE OR LEVEL / TAB ACTION, FIELD SUPP.	PRE-PRINT SKIP / CROSS FOOT	POST-PRINT SKIP / Z/G	SIGN	CHECK DECIMAL	TAB CONDITION OR CONTROL FIELD / INCREMENT	SHIFT TYPE	SHIFT EXTENT	FLOAT CHAR.	MAX. NO. OF CHAR.	LEFT MARGIN	RIGHT MARGIN	PRE-PRINT PROCEDURE / FIELD IMAGE	PRINT CONDITION
1	F	127	N YEAR														
2	F	128	M NAME														
3	F	129	M BONDACCUM														
4	F	130	M ACCGROSS														
5	F	131	M ACCTAX														
6	F	132	M ACCFICA														
7	F	133	M ACCINSUR														
8	F	134	M ACCRET														
9	R	135	42 ERROR														
10	A	136	ERROR-HEADING	HD	EJ	2											
11	F	137	EPAGE						1								
12	A	138	ERROR-LINE	00		1											
13	F	139	EMPLOYNO														
14	F	140	N MONTH														
15	F	141	N DAY														
16	F	142	N YEAR														
17	F	143	ERROR-TYPE														
18	R	144	CHECK														
19	A	145	TITLE-LINE	TT	EJ	ES											
20	A	146	PAYLINE			2											

FORM NO. T1207

REPORT DESCRIPTION FORM

Honeywell
Electronic Data Processing

Title . . . . . Revision . . . . .
Prepared By . . . . . For Program SAMPLE PAYROLL . .
Date . . . . . Checked By . . . . .
Remarks . . . . .

FACT

R	SER. NO.	LINES PER PAGE	REPORT NAME
A	SER. NO.	LINE NAME	
F	SER. NO	FIELD NAME	

	R	SER. NO.	LINES PER PAGE	REPORT NAME	TYPE OR LEVEL	PRE-PRINT SKIP	POST-PRINT SKIP			TAB CONDITION OR CONTROL FIELD						PRE-PRINT PROCEDURE	PRINT CONDITION
	A	SER. NO.		LINE NAME													
	F	SER. NO		FIELD NAME	TAB ACTION	FIELD SUPR.	CROSS FOOT +	CROSS FOOT −	CROSS FOOT =	Z / G	SIGN	CHECK DECIMAL	INCREMENT	SHIFT TYPE	SHIFT EXTENT	FLOAT CHAR.	MAX. NO. OF CHAR.	LEFT MARGIN	RIGHT MARGIN	FIELD IMAGE
1	F	147		EMPLOYNO																
2	F	148		N MONTH																
3	F	149		N DAY																
4	F	150		N YEAR																
5	F	151		TEMPHOURS																
6	F	152		N MONTH																
7	F	153		N DAY																
8	F	154		N YEAR																
9	A	155		PAYLINE						1										
10	F	156		TEMPGROSS		SA														
11	F	157		TEMPTAX				A												
12	F	158		TEMPFICA				A		Z										
13	F	159		RETPRE				A												
14	F	160		INSPREM				A												
15	F	161		BONDEDUCT				A		Z										
16	A	162		PAYLINE					EJ											
17	F	163		NAME																
18	F	164		NETPAY					A											
19	F	165		TEMPGROSS																
20	F	166		NETPAY																

FORM NO. T1207

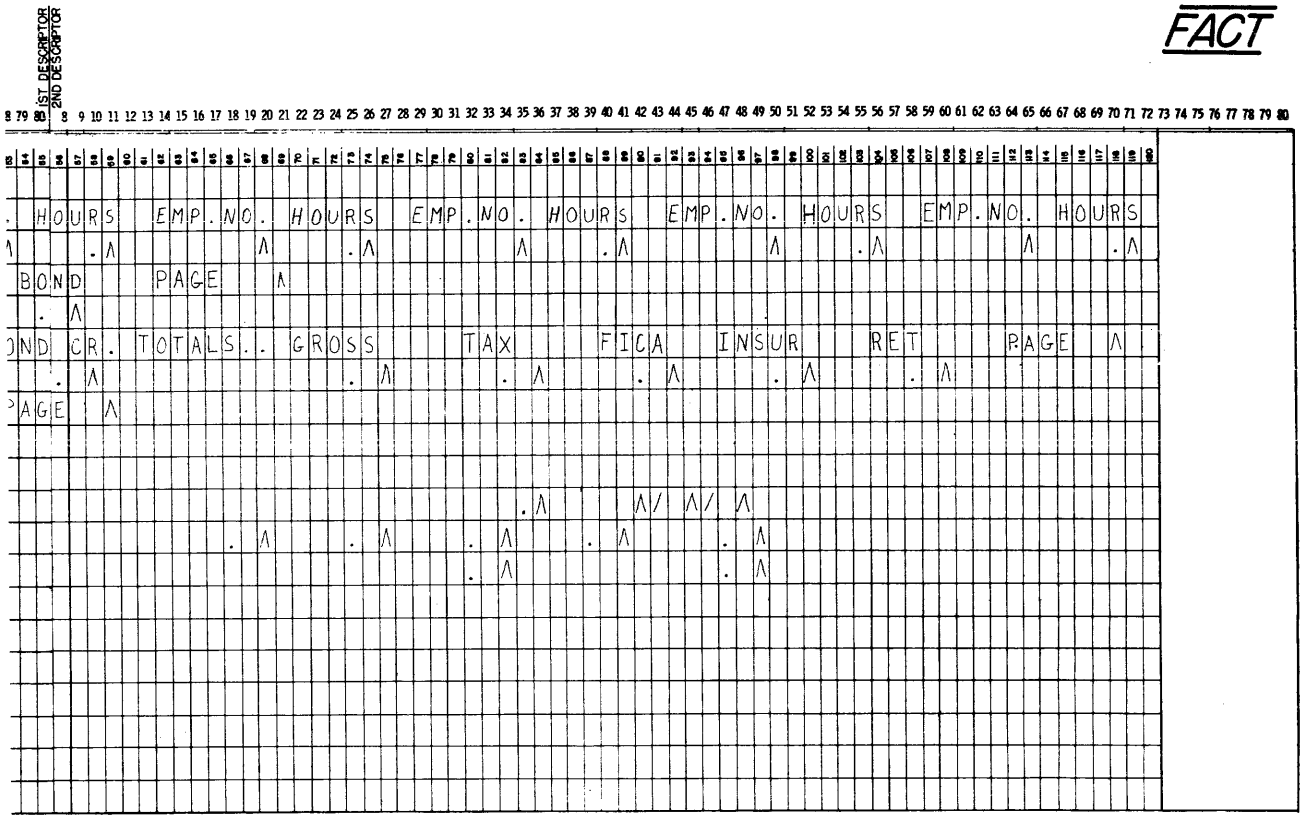
REPORT LAYOUT FORM

Honeywell
Electronic Data Processing

Title . . . . . Revision . . . . .
Prepared By . . . . . For Program SAMPLE PAYROLL . . . . .
Date . . . . . Checked By . . . . .
Remarks . . . . .

L	SER. NO.	SKIP AFTER	LINE NAME	LINE IMAGE
1 L	1		PAGE-HEADING	BATCH NO. Λ IN ERROR PAGE Λ
2 L	2		COLUMN-HEADING	EMP.NO. HOURS EMP.NO. HOURS EMP.NO. HOURS EMP.NC
3 L	3		ERROR-LINE	Λ .Λ Λ .Λ Λ .Λ
4 L	4		BOND-HEADING	BOND ORDERS EMP. NO. NAME DATE
5 L	5		BONDORDER-LINE	Λ Λ Λ- Λ- Λ
6 L	6		DEL-HEADING	TERMINATIONS EMP. NO. DATE NAME B
7 L	7		DELETIONS-LINE	Λ Λ- Λ- Λ Λ
8 L	8		ERROR-HEADING	ERRORS... EMP. NO. DATE TYPE
9 L	9		ERROR-LINE	Λ Λ- Λ- Λ Λ
10 L	10		TITLE-LINE	PLACE CHECK FORM IN PRINTER
11 L	12	1	PAYLINE	Λ Λ/ Λ/ Λ
12 L	13	2	PAYLINE	
13 L	14	3	PAYLINE	Λ $****. Λ
14 L				
15 L				
16 L				
17 L				
18 L				
19 L				
20 L				

FORM NO T1208


FACT
1ST DESCRIPTOR
2ND DESCRIPTOR
HOURS EMP. NO. HOURS EMP. NO. HOURS EMP. NO. HOURS EMP. NO. HOURS
BOND PAGE
ND CR. TOTALS. GROSS TAX FICA INSUR RET PAGE
AGE

SOURCE PROGRAM STATEMENT FORM

Honeywell
Electronic Data Processing

Title . . . . . Revision . . . . .
Prepared By . . . . . For Program SAMPLE.PAYROLL. . .
Date . . . . . Checked By . . . . .
Remarks . . . . .

FACT

	P	SERIAL NUMBER	PROGRAM STATEMENTS
1	P	23	
2	P	24	UPDATED-NEW-MASTER PROCEDURE.
3	P	25	SEE UPDATED-MASTER PROCEDURE.
4	P	26	
5	P	27	NEW-MASTER PRCCEDURE.
6	P	28	IF NEWEMPLOYEE PUT NEWEMPLOYEE AND EMPLOYND INTO PAYROLL-RECORD,
7	P	29	AND PUT ZEROS IN BONDACCUM, ACCGROSS, ACCTAX, ACCINSUR, ACCFICA, AND
8	P	30	ACCRET OF PAYROLL-RECORD. OTHERWISE PUT . . NONEXIST. EMPL . . IN
9	P	31	ERROR-TYPE, WRITE ERROR FROM DETAILFILE, AND DELETE NEW-MASTER ENTRY.
10	P	32	
11	P	33	UNMATCHED-MASTER PROCEDURE.
12	P	34	PUT . . UNMATCHED MASTER. . IN ERROR-TYPE AND WRITE ERROR FROM PR.
13	P	35	
14	P	36	UNORDERED-MASTER PROCEDURE.
15	P	37	PUT . . UNORDERED MASTER. . IN ERROR-TYPE AND WRITE ERROR FROM PR.
16	P	38	
17	P	39	UNORDERED-DETAIL PROCEDURE.
18	P	40	PUT . . UNORDERED DETAIL. . IN ERROR-TYPE AND WRITE ERROR FROM D.
19	P	41	
20	P	42	PROCESS PROCEDURE. IF REGULAR-PAY SEE PAYCALCULATION PROCEDURE AND LEAVE

FORM NO. T1205

Honeywell
Electronic Data Processing

REPORT DESCRIPTION FORM

Title . . . . Revision . . . .
Prepared By . . . . For Program SAMPLE. PAYROLL
Date . . . . Checked By . . . .
Remarks DUPLICATE USING FIELD IMAGE

FACT

	R / A / F	SER. NO.	LINES PER PAGE	REPORT NAME / LINE NAME / FIELD NAME	TYPE OR LEVEL (TAB ACTION / FIELD SUPP.)	PRE-PRINT SKIP (CROSS FOOT + − =)	POST-PRINT SKIP (Z/G)	SIGN	CHECK DECIMAL	INCREMENT	SHIFT (TYPE / EXTENT)	FLOAT CHAR.	MAX. NO. OF CHAR.	LEFT MARGIN	RIGHT MARGIN	FIELD IMAGE (PRE-PRINT PROCEDURE)	PRINT CONDITION
1	R	1	40	ERROR-REPORT													
2	A	2		PAGE-HEADING	HD	EJ	2										
3	F	3		BATCH-NUMBER											24	BATCH NO. ^	
4	F	4		PAGE-NUMBER						1					43	IN ERROR PAGE ^	
5	A	5		COLUMN-HEADINGS	HD		3										
6	F	6													30	EMP. NO. HOURS EMP. NO. HOURS ^	
7	F	7													60	EMP. NO. HOURS EMP. NO. HOURS ^	
8	F	8													90	EMP. NO. HOURS EMP. NO. HOURS ^	
9	F	9													120	EMP. NO. HOURS EMP. NO. HOURS ^	
10	A	10		ERROR-LINE	00		1										
11	F	11		1ST EN											8	^	
12	F	12		1ST RH											14	.^	
13	F	13		2ND EN											23	^	
14	F	14		2ND RH											29	.^	
15	F	15		3RD EN											38	^	
16	F	16		3RD RH											44	.^	
17	F	17		4TH EN											53	^	
18	F	18		4TH RH											59	.^	
19	F	19		5TH EN											68	^	
20	F	20		5TH RH											74	.^	

FORM NO. T1207

REPORT DESCRIPTION FORM

Honeywell
Electronic Data Processing

FACT

Title .... Revision ....
Prepared By .... For Program SAMPLE PAYROLL
Date .... Checked By ....
Remarks DUPLICATE USING FIELD IMAGE

	R SER. NO.	LINES PER PAGE	REPORT NAME			
	A SER. NO.	LINE NAME	TYPE OR LEVEL / PRE-PRINT SKIP / POST-PRINT SKIP	TAB CONDITION OR CONTROL FIELD	PRE-PRINT PROCEDURE	PRINT CONDITION
	F SER. NO	FIELD NAME	TAB ACTION / FIELD SUPP. / CROSS FOOT + − = / Z/G / SIGN / CHECK DECIMAL / INCREMENT / SHIFT TYPE EXTENT / FLOAT CHAR. / MAX. NO. OF CHAR. / LEFT MARGIN	RIGHT MARGIN	FIELD IMAGE	
1	F 21	6TH EN		83	^	
2	F 22	6TH RH		89	.^	
3	F 23	7TH EN		98	^	
4	F 24	7TH RH		104	.^	
5	F 25	8TH EN		113	^	
6	F 26	8TH RH		119	.^	
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						
19						
20						

FORM NO. T1207

FACT LEXICON

ADD (verb)
AND (conj.)
ARE (verb)
BLANK (adj.)
BY (prep.)
CHARACTER (noun)
CLOSE (verb)
CONTINUED (adj.)
CONTROL (verb, noun)
DELETE (verb)
DIGIT (noun)
DIVIDE (verb)
DIVIDED (adj.)
DO (verb)
EACH (adj.)
EITHER (conj.)
END (noun)
ENTRY (noun)
EQUAL (adj.)
EQUALS (verb)
ERROR (noun)
EVERY (adj.)
EXCEPT (conj.)
EXECUTE (verb)
FIELD (noun)
FIELDS (noun)
FILE (verb, noun)
FROM (prep.)
GET (verb)
GO (verb)
GREATER (adj.)
GROUP (noun)
IF (conj.)
IN (prep.)
INTO (prep.)
INVALID (adj.)
IS (verb)
LEAVE (verb)
LESS (adj.)
LETTER (noun)
LINE (noun)
MATCHED-MASTER (adj.)
MATCHED-NEW-MASTER (adj.)
MINUS (prep.)
MULTIPLIED (adj.)
MULTIPLY (verb)
NEITHER (conj.)
NEW (adj.)
NEW-MASTER (adj.)
NEXT (adj.)

Table I

NO (adj.)
NONE (noun)
NOR (conj.)
NOT (adv.)
NOTE (noun)
OF (prep.)
ON (prep.)
OPEN (verb)
OR (conj.)
OTHERWISE (conj.)
OVER (prep.)
PAGE (noun)
PARAGRAPH (noun)
PLUS (prep.)
POSITION (noun)
PRIMARY (adj.)
PROCEDURE (noun)
PUT (verb)
REMAINDER (noun)
REPLACE (verb)
REPORT (noun)
RETURN (verb)
REVERSE (verb, adv.)
SEE (verb)
SET (verb)
SKIP (verb)
SORT (verb)
SUBTRACT (verb)
TABLE (noun)
THE (adj.)
THEN (adv.)
TIMES (prep.)
THROUGH (prep.)
THRU (prep.)
TO (prep.)
UNCHECKED (adj.)
UNLESS (conj.)
UNMATCHED-MASTER (adj.)
UNORDERED-DETAIL (adj.)
UNORDERED-MASTER (adj.)
UNTIL (prep.)
UPDATED-MASTER (adj.)
UPDATE (verb)
UPDATED-NEW-MASTER (adj.)
VALID (adj.)
VALIDITY (adj.)
WHEN (conj.)
WITHIN (prep.)
ZERO (noun)
ZEROES (noun)
ZEROS (noun)

Table I (continued)

Honeywell

Electronic Data Processing