IBM

IBM 1401 Data Processing System Programmers' Guide

Student Text

IBM

Student Text

IBM 1401 Data Processing System

Programmers' Guide

Preface

The *IBM 1401 Programmers' Guide* is intended for experienced operators of IBM unit record equipment who want to learn how to program the IBM 1401 Data Processing System.

A number of technical manuals and bulletins on various phases of the system are available. Most of them, however, presuppose a knowledge of programming techniques.

The purpose of this guide is to present programming principles in a sequence and language that makes learning easy for the beginner. The subjects are presented in a step-by-step fashion to help the beginner learn the operations a few at a time, and apply them immediately to a problem. Whenever possible, 1401 technical terms are associated with situations familiar to the reader.

The guide has three main parts:

Part 1: Introduction to Data Processing
Part 2: Problems and Solutions
Part 3: Introduction to Tape Records

The information in *Part 1* is written primarily for those readers who feel the need for some background material before tackling the problems presented in *Part 2*.

The eight problems in *Part 2* involve punched card records and files normally encountered in a payroll application. In processing data contained in these records, practically all of the 1401 operation codes associated with standard features and a few special features (such as multiply-divide, expanded print edit, and sense switches) are used in the solution of the eight problems. The problems range from simple reproduction of a card file to the preparation of payroll registers, payroll checks and earnings statements. Problems 1, 2, and 3 are programmed in actual machine language. Problems 3, 4, 5, 6, 7, and 8 are programmed in symbolic language.

Part 3 presents a problem involving the updating of a master tape file. The problem is solved in three different ways. The first two solutions involve two different symbolic program systems (SPS and *Autocoder*). The third solution involves the use of an input/output control system (IOCS) in conjunction with *Autocoder*. IOCS eliminates the need for detailed programming of standardized input and output routines.

Because these problems are well within the range of even the smallest 1401 system, no attempt is made to conserve storage, employ shortcuts, or use sophisticated approaches to the problems. The main emphasis is on ways and means of tackling comparatively simple but practical problems in a straightforward manner, more suited to the beginner than to the experienced programmer.

To get the most out of this guide, use it in conjunction with the following IBM publications:

IBM 1401 Data Processing System Reference Manual, A24-1403.
IBM 1401 Symbolic Programming Systems, C24-1480.
IBM 1401 Autocoder Specifications, J24-1434.
IBM 1401 Input/Output Control System, C24-1462.

This guide refers to these publications as the need for detailed technical information arises.

MINOR REVISION (March 1965)
This edition, C24-1494-2, is a minor revision of, but does not obsolete, C24-1494-0 and C24-1494-1.

Copies of this and other IBM publications can be obtained through IBM branch offices. Address comments concerning the contents of this publication to IBM, Technical Publications Department, 112 East Post Road, White Plains, N.Y. 10601

Contents

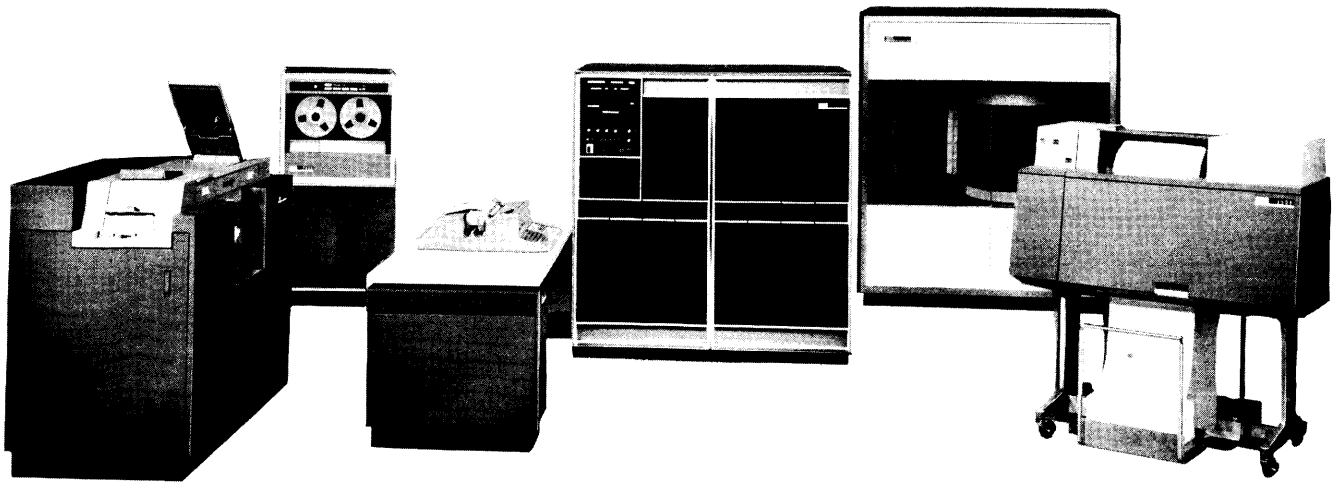


Figure 1. IBM 1401 Data Processing System

Part 1. Introduction to Data Processing

For those of you who have wired control panels on IBM unit record equipment, the IBM 1401 is a reproducer, a gang punch, a summary punch, a calculator, an accounting machine, and a collator (some limited features of) – all rolled into one. The 1401, therefore, is regarded as a data processing system rather than a machine.

In the processing of data contained in punched cards, all of the functions normally performed by these individual machines with control-panel wiring can be performed by the 1401 in a single pass of the cards with instructions that you, as a programmer, have to develop.

In data processing, the desired result may be a management report, a tax report to the Social Security Board, a financial statement, an inventory report, a payroll register, a check and earnings statement, etc. Or it may be the solution to a complicated scientific formula that involves literally millions of machine calculations.

In a broader sense, computers are used to guide missiles, track satellites circling in outer space, and give information that tells exactly where the missile or satellite is at any given moment of time, even though it may be traveling thousands of miles an hour.

Through the use of computers, scientists and engineers can "fly" a jet plane on paper long before it ever reaches the drawing board stage. Information to be processed is fed into the computer with cards, paper and magnetic tape, and (very recently) over telephone wires from locations thousands of miles away. IBM TELE-PROCESSING® Systems have opened a whole new field to the data processing industry, and the surface is just barely scratched. Experiments are being conducted that will eventually permit people to communicate with the computer with voice sounds. The tremendous progress in the physical sciences in just the last decade would not have been possible without computers. What the future holds forth is anybody's guess.

Data Processing

As an experienced operator of IBM unit record equipment, you are familiar with the term *data processing*. But has it ever occurred to you that you have processed data without a machine from the time you were able to use a pencil and paper? You process data whenever you:

- Add one figure to another to get the sum, or subtract one figure from another to get the difference.
- Multiply one figure by another to get the product, or divide one figure by another to get the quotient.
- Add a column of figures on a sheet of paper.
- Compare one record with another, either in whole or in part.
- Decide to take a different course of action based on results obtained in a previous operation.
- Copy information from one record, in whole or in part, and write it on another.
- Take a file of records (a collection or group of similar records) that are in random order and arrange them in a predetermined sequence, numerically or alphabetically.
- Take two or more related files that are in the same order, and combine them into one file.
- Search through a file of records to do something with the record or records selected.
- Change information in a record to correct or update it.
- Put the record back into the file where it belongs.

Even when you take a file of records and put it away in storage for later use, you are processing data. Generally, anything you normally do with records without a machine, you can do with a computer *if the records are in a form that a computer can read.*

Data processing, therefore, is basically reading information contained in records, and doing something with it.

Now, go back to the list and read it again. This time visualize yourself doing these operations manually. Take careful note of your mental processes involved. With some operations, you jot down the figures you want to work with on a piece of paper, and develop the answers on the same, or possibly another, piece of paper. With others, it may not be necessary to do any writing. You rely on your memory instead.

Some operations require a knowledge of arithmetic. Others involve logic. For example, deciding what to do when two numbers are compared is a logical operation. If they are equal, you may take one course of action. If they are unequal, you may take another. You may even have to decide what to do if one number is lower (or higher) than another.

Another example of logic is based on the result of an arithmetic operation. Subtract one figure from another. If the result is zero, continue with the next sequential operation. If the result is plus or minus, skip some portion of the program, go (branch) to another portion, and then continue.

The point is that whether you write things down or keep them in your head, you use some storage device to store not only what you want to do (program) but also the basic data you are working with (input) and the results you expect to produce (output).

Programming

Learning a technical subject like programming a computer cannot be done overnight. Learning of any kind is a gradual accumulation of knowledge and experiences. With each bit of knowledge gained, with each new experience, we store a reservoir of information in our brain that gives us confidence and courage to make decisions and go further ahead. We know where we are going and why.

Consider how a child learns to walk. First, he *creeps*. Then he *stands up*, pulling himself up by the bootstrap as the saying goes. He then decides to *move* his legs and *take the first step*. Within a short time, he starts to *walk* not just a *straight line*, but to *turn corners*, to *come back*, and to *repeat*. Before long he is running instead of walking.

Now, list the words in italics in the order they appear. You have essentially what in data processing terminology is a *program*. Programmers will recognize this program as a *closed loop*, because the steps in the program are repeated without deviation as often as required. The word bootstrap is not in italics but programmers often start executing a program with a bootstrap instruction.

A program, therefore, is a series of instructions that when successfully carried out (executed), accomplishes the desired result.

Stored Program

The IBM 1401 is a *stored program* data processing system. This simply means that the program you develop to solve a problem must be stored somewhere in the computer in order to execute it. You first write the program on specially designed code sheets from which instruction cards are punched. A *program loading routine* then enters (loads) the instructions into the storage unit. The storage unit (the technical name is *processing unit*), has other names such as *logical unit*, *memory*, and *core storage*.

Once the program is loaded into the processing unit, the records you want to process must be fed into *work areas* of the same storage unit by means of punched cards or magnetic tape. After each record is processed, the machine can print the results on blank paper or on specially designed report forms, one line at a time. Or if you choose, you can punch the results in other punched cards or write them on magnetic tape for later processing.

Figure 2 shows schematically the interrelationship between stored program, input data, processing, and output data.

Programming Language

The 1401 can execute a program only when it receives instructions in a language it can understand. This language is known as *actual* or *absolute machine language*.

Originally, all programs were written in actual machine language. But writing programs with actual

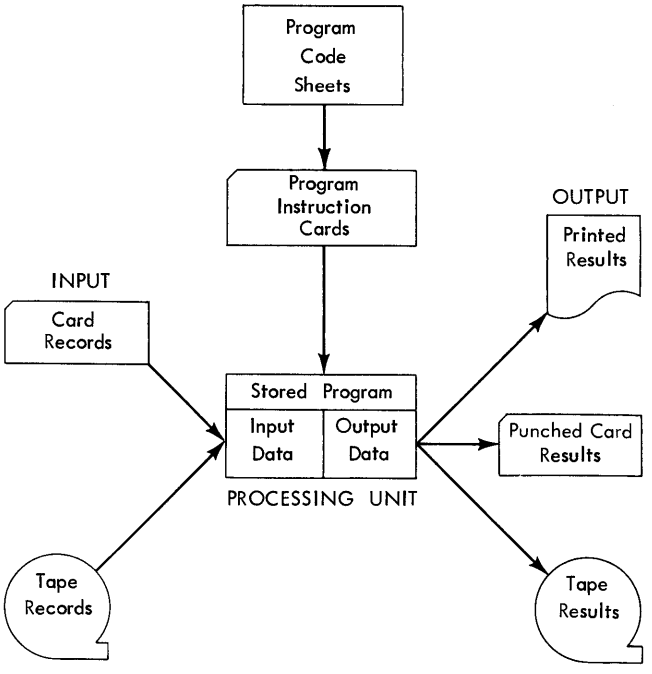


Figure 2. Schematic of the Components of the IBM 1401 Data Processing System

addresses creates problems in assigning data to storage locations, makes it difficult to write cross references within a program, leads to difficulties when several people must work on the same job, and generally results in programs that are difficult to correct and to modify.

For these reasons, almost all actual programming is done with a symbolic programming system. For the 1401 system there are three rather similar symbolic programming systems available: SPS-1, SPS-2, and *Autocoder*. Programs written in symbolic language are *source programs*. Before source programs can be used, they must be translated to actual machine language. Special programs known as *processors* make the translation from symbolic to actual. Source programs translated to actual machine language are *object programs*.

We shall learn more about symbolic programming later on after we program a few problems in actual machine language.

When we use actual machine language, each instruction that a 1401 computer can execute is represented by a number, a letter, or a special symbol. Practically all of the characters you find on a standard typewriter are used as codes to denote a specific type of operation.

For example: 1 stands for READ A CARD, 2 for WRITE A LINE, 3 for WRITE A LINE AND READ A CARD, 4 for PUNCH A CARD, 7 for WRITE-READ-PUNCH.

These codes are classified as *input-output* codes because they deal with the source information read into the system (input), or with the results produced by the system (output). Thus a card reader is an input unit, and a card punch is an output unit. A printer is an output unit because it produces a printed report or document in another form. Another type of input or output is a tape unit.

Other codes are classified as *arithmetic codes*. A stands for ADD, S for SUBTRACT, @ for MULTIPLY, and % for DIVIDE.

In addition to the machine language codes mentioned, there are *logic operation* codes, *move and load* codes, and *miscellaneous operation* codes. The important thing to remember at this point is that operation codes, regardless of type, are represented by a single character. You can easily memorize most of them, particularly those used most frequently. Code charts are available for quick and easy reference, when you are writing a program in actual machine language.

Records

Records are basic management tools without which a business could not exist. In modern business, records come in a variety of forms, sizes and colors, but regardless of these variables, they contain information about people, things, or events of one kind or another.

Some records are required by law. Others are created as they are needed in the normal course of human activity. For the most part, information regarding people, things, or events is first recorded on paper forms designed for specific purposes.

A personnel record contains information about a person, and identifies the person by number (in most cases) as well as by name. This information usually comes from an application form, which the applicant for a job filled out.

A payroll record contains information about an employee's earnings. The basis for this information is a time card or an attendance record.

A check contains information (date, name, amount) about an event (payment of a bill or for services rendered). The deposit slip you make out when you put money in a bank, or the check you write when you withdraw money, are records.

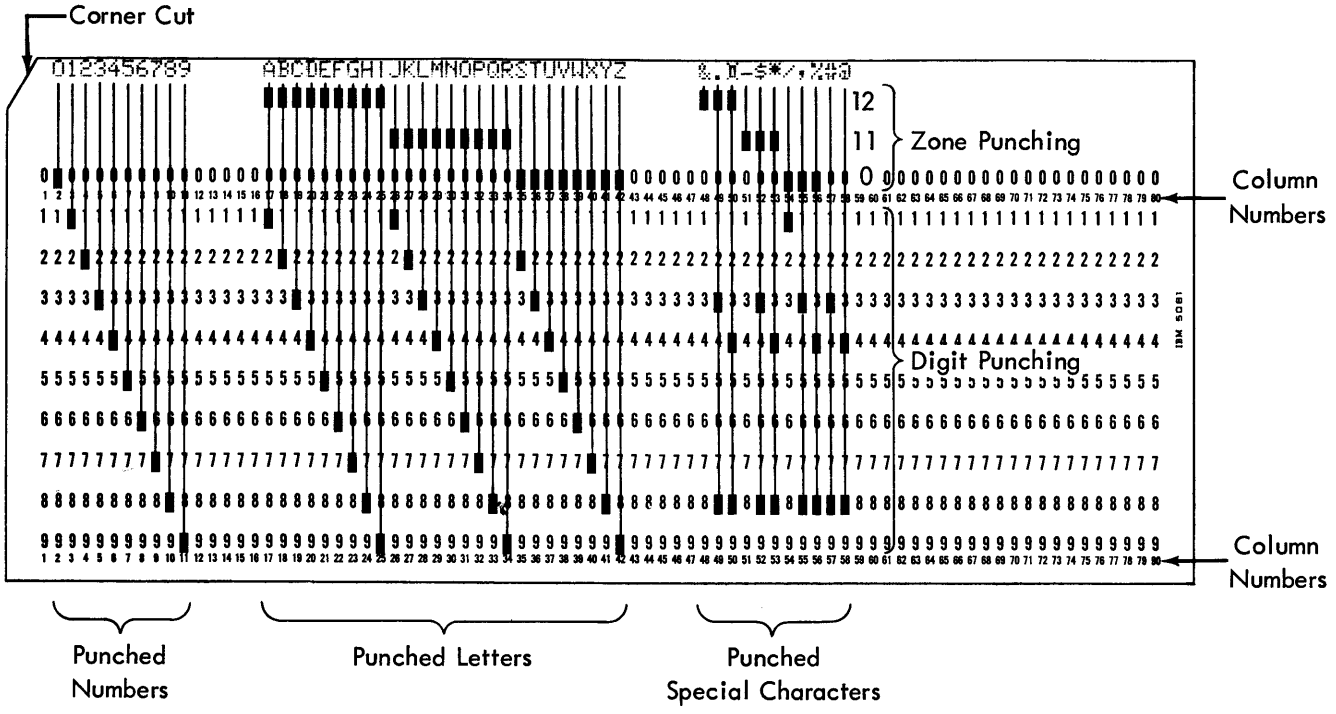
The sales slip that a clerk makes out when you buy something in a store for cash or on credit is a record. If you pay cash, your copy of the sales slip is marked PAID and is your record of the transaction. If you buy on credit, the amount is charged to your account. Some time later, a statement (another record) will surely be in the mail.

You see, once a transaction or an event becomes a matter of record, processing must follow as surely as night follows day. Otherwise, there is no need for the record in the first place.

The handling of these records speedily, accurately, and economically is the main purpose of data processing equipment. But before these records can be processed with data processing equipment, the information they contain must be presented in a machine-readable language in the form of punched holes in cards or magnetic spots (bits) on tape. Various combinations of holes in cards or bits on tape represent numbers 0-9, letters A-Z, and eleven special characters: &.□-$*/,% # @

Punched Card Characters

Figure 3 shows what these characters look like when punched in a card. Note that the capacity of the card is 80 positions (columns). In column 7, the punched hole stands for the digit 5. In column 10, the punched hole stands for the digit 8. In column 24, the combination of a 12-punch and an 8-punch represents the letter H. And in column 33, the combination 11-8 means the letter Q. In column 41, the combination 0-8 stands for the letter Y. Note further that the alphabet is divided into three sections called *zones*. The letters A-I fall into the first section (12-zone), J-R into the second section (11-zone), and S-Z into the third section (0-zone).



Note: This figure is a reduced reproduction of a standard card, not designed for any specific purpose. Actual dimensions are 7 3/8 inches from left edge to right edge, and 3 1/4 inches from top edge to bottom edge. Corner cut may be specified for either upper left or upper right.

Figure 3. IBM Card-Codes

A zone punch, therefore, in combination with any one of the digits 1-9 represents a specific letter of the alphabet. The letters H, Q, and Y all have in common the digit punch 8. The difference is that they all have different zone punching (12 for H, 11 for Q, and 0 for Y). An 8-punch without a zone punch is simply the digit 8. Other combinations of zone and digit punching represent special characters as shown in columns 48 through 58. The virgule, or slash, (/) is the combination of 0-1. The comma (,) is the combination of 0-3-8.

All the characters shown in Figure 3 can be printed. Later on we will come across other special characters that cannot be printed, but are used as operation codes. The most common of these are the question mark (?) and the exclamation point (!), represented in punched card form as 12-0 and 11-0 respectively.

Magnetic Tape Characters

Figure 4 shows the bit combinations for these characters when written on magnetic tape. Each bit is a magnetized spot on the tape that is sensed electronically. They are not visible to the naked eye.

In comparing the characters in punched hole form with the characters in bit form, note that each vertical column of the card in Figure 3 has 12 possible punching positions whereas each vertical column of the tape has only 7 possible bit positions. Also note that there are no bit positions by themselves that stand for 0, 3, 5, 6, 7, or 9. These digits are represented in punched cards by a single hole.

On tape, they are represented by a combination of bits designated 1, 2, 4, and 8. Disregarding the C- or check bit for a moment, bits 2 and 8 stand for 0, bits 1 and 2 for 3, bits 1 and 4 for 5, bits 2 and 4 for 6, bits 1, 2, and 4 for 7, and bits 1 and 8 for 9.

This pattern of bits is called the *binary coded decimal* system, abbreviated as BCD. The word *bit* derives from the first letter in the word *b*inary and the last two letters in the word dig*it*. The BCD system is also used to represent characters in the storage device.

Letters and special characters are represented by a combination of numerical bits (8421) and zone bits (BA). B- and A-bits, in combination, correspond to the 12-zone punch. The B-bit corresponds to the 11-zone punch, and the A-bit to the 0-zone punch. The letter C, for example, which is the third letter in the 12-zone of the alphabet (card code 12-3), is a combination of BA21 bits. BA is the same as 12, and 21 is the same as 3.

This covers six of the seven possible bits that are used to represent a character. The seventh bit (C) along the top row of the tape is a built-in checking feature that the computer automatically supplies when characters are written on tape.

Note that the check bit is not part of the character configuration when the number of BA8421 bits that represent the character is even. It appears only for those characters where the number of bits BA8421 is odd. The automatic inclusion of the check bit (also called *parity bit*, or *redundancy bit*) changes the configuration of the character from an odd number of bits to an even number of bits. Thus, all characters shown in Figure 4 are written in the *even-parity mode.*

Whenever a character on tape has an odd number of bits, the character is invalid, and a tape error is indicated (the machine stops). This can happen due to an imperfection in the tape or to dust particles, or in rare cases to a machine malfunction. A record that contains one or more invalid characters is regarded as unreadable.

Some tape systems write tape in the *odd-parity mode.* To make the 1401 compatible with other tape systems, special instructions are available for either reading or writing tape records in the odd-parity mode.

Words and Numbers

Up to this point, we have learned that digits, letters and special characters can be recorded in a form that machines can read. It should not surprise anyone then that machines can read and/or write words or numbers of more than one character, because a word is a combination of letters and a number is a combination of digits.

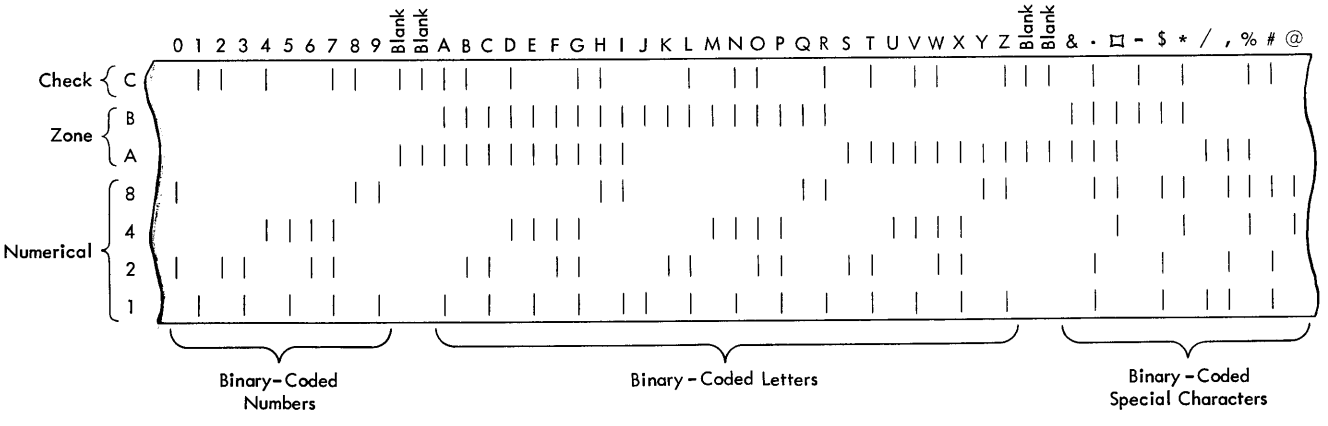
The term *word,* we normally associate with letters of the alphabet. Programmers, however, also use it to denote a number, even though the number may consist of a single digit. Thus, all items or units of information that make up a record are called *words.*

When all the words in a record have the same number of characters, they are *fixed-length words.* When all the records have the same number of characters, they are *fixed-length records.* When they are not fixed, they are variable. Thus, we have *variable-length words* and *variable-length records.*

A 1401 instruction, which may consist of a single character but may have as many as eight, is a *variable-word-length instruction.* This is an important feature of the 1401 because some computers handle only fixed-length words. Early computers handled only ten-character words. This resulted in a considerable waste of storage capacity for words of less than ten characters. If a word had only two characters (such as, 89), eight storage positions were wasted.

Calling a number a word may be confusing to you. But it should be no more confusing than referring to an item of information in a punched card as a *field. Words* and *fields* are synonymous terms. They both are defined as: *a character or set of characters treated as a single unit of information.*

Field applies more specifically to punched cards where each variable-length field is separated from all others with vertical lines. *Word* applies generally to 1401 storage where each variable-length word is set apart from all others with a *word mark.* Thus, vertical lines on punched cards and word marks in storage serve the same purpose.



Note: This figure is an enlarged cutaway illustration of a section of magnetic tape. Actual magnetic tape dimensions are one-half inch wide and 2400 feet long. The tape is wound on a reel 10 1/2 inches in diameter. Tape can accommodate 200 characters per inch when written in low density, and 556 characters per inch when written in high density. Over 14,000,000 characters can be written on one reel of tape. The end of one tape record and the beginning of the next is indicated by a three-quarters inch gap (inter-record-gap).

Figure 4. IBM Tape-Codes

In storage, word marks are indicated with a special word-mark bit. In text, they are indicated by underscoring the

firstcharacterofeachwordornumber123896720.

The first character of a word is usually called the high-order (leftmost) character. The last character of a word is usually called the low-order (rightmost) character. In the word 123896720, the 1 is the high-order character and the 0 is the low-order character.

Words that are made up of letters of the alphabet are, for the most part, self-explanatory. But words that are made up of digits are by themselves meaningless. Take the number 123896720. It could represent the number of miles from one planet to another. Since there are nine digits in the number, you could guess that it's a social security number because all social security numbers have nine digits. You would be wrong because social security numbers are really three numbers separated by two hyphens 123-89-6720. Assume you were right. Then the question arises, "Whose social security number?"

Now, add a dollar sign, a couple of commas, and a decimal point in the proper places so that it becomes $1,238,967.20. And there is no question but that it stands for dollars and cents.

Incidentally, when a programmer says that he is going to *edit* an amount, he means that he is going to punctuate it with special characters. But even though we have clarified the number to some extent, there is still something missing. Does it mean that the company has that much money in the bank? If so, what bank? Does it represent the value of a certain piece of property owned by the company? What property? Or does it mean that the company owes that much money to one of its suppliers? What supplier?

We could go on indefinitely with questions of this type and get nowhere. The point is that a number must have some identification (some label attached to it) in order to make it meaningful.

Data Words, Constant Words, and Instruction Words

Words processed by the 1401 fall into three general categories.

Data Words. These are the words that make up the records. Although most of these words are numbers, they are all identified with labels of one kind or another.

Constant Words. These are the words that do not appear in your input records but are needed to print repetitive descriptive information for a report. A constant word may simply be a number needed to perform some multiplying, dividing, or adding operation. For example, counting records is normally done by adding a constant 1 for each record processed.

Instruction Words. These tell the computer what to do, and when to do it. Instruction words may vary from one to eight characters, and the first character signifies an operation.

Code Numbers

In data processing (in fact, in all forms of record keeping) numbers are used not only in the quantitative sense, but also as codes to represent a piece of information that is ordinarily described with a name or label. Instead of writing: *October thirteen, nineteen sixty-two,* we write: *10/13/62.* Month (10), day (13), and year (1962) are implied. Similarly, in a payroll record, instead of saying: *Daniel Glenn, in the Products Publication Department, worked forty hours during the week ending on that date, at the rate of three dollars an hour, and therefore earned one hundred and twenty dollars,* we can record the information on a payroll record form (properly designed) by simply using numbers (Figure 5).

The sample payroll record in Figure 5 contains six variable-length words. All are numbers which wouldn't mean a thing without the labels to identify them. The number 114570 is Daniel Glenn's code number. 171 is the code number for Products Publication Department. It is plain to see that by using code numbers instead of names, we save ourselves a lot of writing, and incidentally a lot of processing. For instance, we could develop a simple code list for the 52 weeks of the year, assign a two-digit number to each week, give it a name like ROLL NUMBER, and record 41 instead of 10/13/62.

In actual practice, weekly payroll records are most often designed as *clock cards,* and placed in a rack located near a time clock. During the week, employees use the clock to register their INS and OUTS for each day. At the end of the week, a time clerk calculates the total hours worked, and writes it on the card. The accounting department then calculates the amount earned.

PAYROLL RECORD

Week Ending	10/13/62
Man Number	114570
Department Number	171
Hours Worked	40
Hourly Rate	3.00
Amount Earned	120.00

Figure 5. Payroll Record

Computing Tax Deductions and Net Pay

We know very well, of course, that Daniel Glenn isn't going to get a check for $120 for his week's work. The social security tax (F.I.C.A.), federal withholding tax, and sometimes state taxes have to be computed and deducted. Moreover, figuring these taxes on the basis of $120 would be a waste of time, if we didn't first make sure that $3 times 40 hours equalled $120.

In figuring the social security tax, we must keep in mind that $150 is the limit that any one person has to pay for social security. This is figured at the present rate of 3⅛% of the first $4,800 earned during the year. During the first few months of the year, he undoubtedly has to pay the social security tax (120 × 3⅛%) but by the time the 41st week rolls around (10/13/62), he has reached the $150 limit, and therefore pays no more social security tax for the rest of the year. If what he has paid in for the year-to-date is within a few dollars or a few cents short of the maximum amount, he only has to pay the difference and not the full amount. Example:

Maximum social security tax	$150.00
Deductions as of 10/06/62	148.23
Social security tax for week ending 10/13/62	$ 1.77

Withholding tax is based not on gross pay but on taxable earnings. This is the difference between gross pay and allowances for dependents. For every person claimed as a dependent, including himself, $13 (weekly rate) is deducted from his weekly pay before the withholding tax is computed. At the present time, this is computed at the rate of 18% on taxable earnings. Without dependents, his tax is computed on $120. With one dependent, his tax is computed on $107 ($120 −$13), etc.

So you see, in order to make out the check to Daniel Glenn in the right amount, we must take into consideration not only his earnings and tax deductions for the current week, but for the year-to-date. If he has authorized deductions for community chest, hospital insurance, U.S. Savings Bonds, etc., the amounts authorized must be deducted. With all these deductions from his earnings of $120, Daniel Glenn would consider himself a pretty lucky fellow with take-home pay of $89.32.

This gives you a rough idea of what is involved in just one problem that every company faces when preparing a payroll and writing checks.

We shall do this problem in *Part 2. Problems and Solutions.*

Card Design

Before the payroll record shown in Figure 5 can be processed with data processing equipment, the data it contains must first be transcribed onto cards in the form of punched holes. To facilitate punching of the data, a payroll card form must be designed. The items of information the card form must provide are:

Week ending	10/13/62
Man number	114570
Department number	171
Hours worked	40
Hourly rate	3.00
Amount earned	120.00

In designing the card, the first thing we have to decide is: *How many columns do we need for each item or field?*

Week Ending, as written, requires eight columns, if we include the slashes. But the slashes can be assumed. So we can cut it down to six. We can also eliminate the tens position of the year (6) because it will be the same for the next eight years. This reduces the number of columns to five. If the record is of a temporary nature, we could easily eliminate the year altogether, and cut it down to four columns.

Because we are trying to eliminate as much recording as possible without destroying its significance, we can decide to use the 2-digit roll number instead. Thus, columns 1 and 2 are reserved for recording the roll number, which we will identify with a field heading, in other words, a descriptive label. Draw a vertical line between columns 2 and 3, and proceed with the next field of information.

Man Number requires six columns, if we are sure that we will never have to record a man number higher than 999999. So, columns 3-8 (properly identified by a field heading) are reserved for recording man number. Draw a line between columns 8 and 9, and proceed with the next field.

These vertical lines on the card indicate the end of one field and the beginning of the next. The position to the left of the line is the *low-order* position of one field, and the position to the right of the line is the *high-order* position of the next field.

Department Number, as written, normally requires three columns. But we know that the highest department number is 2925. Therefore, we need four columns for some numbers. This brings us out to column 12.

Hours Worked, as written, requires two columns, if we assume that no one will ever work more than 99 hours in a week. But we know that sometimes our employees work fractions of hours (expressed as a one-position decimal figure). (*Example:* 38.5 or 36.7). We therefore need three columns, if we assume the decimal point. Although we do not punch the decimal point, we can easily indicate it on the card by drawing a broken vertical line between columns 14 and 15. If we want the 1401 to print the decimal point where it belongs, we can always do it in our program with an EDIT instruction.

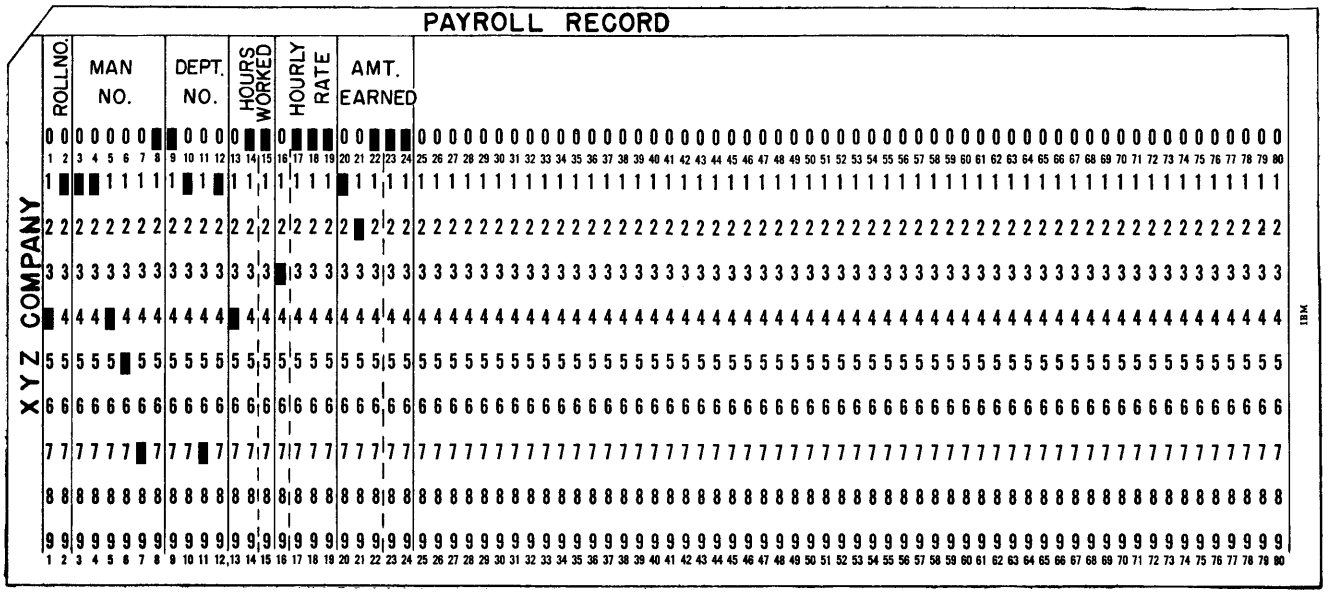


Figure 6. Payroll Record Card Form

Hourly Rate, as written, requires three columns, if we assume the decimal point. But we know that some of our rates have three decimal positions. (*Example:* 1.275). This practice is not uncommon in some companies. So we need four columns, and this brings us out to column 19. While we are about it, let's not forget the broken dotted line to indicate the decimal point.

Following the same rules, we proceed with the *Amount Earned* field in columns 20 to 24. The card that an operator would punch for the payroll record shown in Figure 5 would look like Figure 6.

Types of Card Records

Admittedly, Figure 6 was a simple card form to design. Everything seemed to fall into place without any confusion, simply because we followed a few rules, and patterned our design in the same sequence as the basic payroll record (Figure 5). In practice the job of designing cards is not so easy, particularly when a job involves the processing of cards with two or more different designs.

For example, a typical payroll application usually requires, in addition to the current payroll record card, an employee name card, a year-to-date earnings card, and a miscellaneous deductions card. Sometimes these cards are processed as individual files. Other times, they are merged into one file and processed. Here is where experience in systems planning pays dividends. Poorly-designed forms can create problems that may be difficult (or impossible) to handle. Systems planners foresee possible trouble areas and take steps at least to minimize the difficulty, if not to eliminate it altogether.

One thing we can be sure of. When two or more card designs are involved in a given job, each type of card must be distinguished from all other types by a code in some selected column of the card. The most common method of distinguishing two types of cards is: punch an X (11-punch) in some selected column of one card, and do not punch an X in the other type. Do not confuse this X with the letter X. Thus, we may say that employees' name cards are X80, for example, and current payroll record cards are NoX80. Quite often, even though only one card design is used, types of transaction cards are distinguished by different combinations of two or more X-punches.

Four different types of payroll deductions could be distinguished by using two X's:

Type of Deduction	*Col. 79*	*Col. 80*
Community chest	X	
Hospital insurance		X
Union dues	X	X
Bonds		

Use a third X, say in column 78, and you can easily add four more combinations, each representing another type of deduction. You can get into all sorts of combinations to distinguish cards by type by using digits, letters, or special symbols.

Reading Card Data

The punched card shown in Figure 6, without regard to field headings, is simply one long string of 24 digits: 411145700171400300012000. By itself, it is meaningless. It is only when we break this number down with vertical lines or word marks:

41|114570|0171|400|3000|12000|
41 114570 0171 400 3000 12000_

that we realize it is not one but six numbers. In order to understand what these numbers represent we have to refer to the six field headings: *Roll No., Man No., Dept. No., Hours Worked, Hourly Rate,* and *Amount Earned.*

Data processing equipment cannot read field headings or determine for itself that the 24 digits represent six words of variable lengths. Therefore, it is up to the programmer to supply this information to the machine, one way or another, and to handle it accordingly.

With unit record equipment, field assignments are a matter of the correct use of wires on the control panel. With the 1401 (which has no control panel), field assignments are handled by SET WORD MARK instructions. The 24 digits punched in the card in Figure 6 would appear in storage in the BCD (odd-parity) mode with a word-mark bit (eighth bit) under the high-order position of each word. The bit combinations for the first twelve digits in the example 411145700171 are shown in Figure 7. The 1's denote bits and the 0's denote no bit.

The complete table of characters, with their BCD codes and card codes, is shown in Figure 8. In Figure 7, the bit configurations are shown vertically. In Figure 8, they are shown horizontally. Notice that wherever a word mark is part of the configuration, odd parity is maintained by adding or removing a check bit.

Storage Units

Before we discuss storage units in computers, let's mention a few that you are familiar with. A tape recorder stores sounds on magnetic tape. Other recorders store sounds in grooves on rubber or aluminum discs. Videotape stores moving pictures as well as sounds, and synchronizes the sounds with the action displayed on your television screen.

Storage units in computers store information represented by digits, letters, and special characters. Broadly speaking, a punched card is a storage device of one record, and a card file is a storage device of many records. Magnetic tape reels, or magnetic disks, are storage devices of many records. Because, by themselves, cards, reels, and disks are not a part of the computer, they are often referred to as *external storage.* It is only when the information they contain enters into an *internal storage* device inside the computer that the records can be processed.

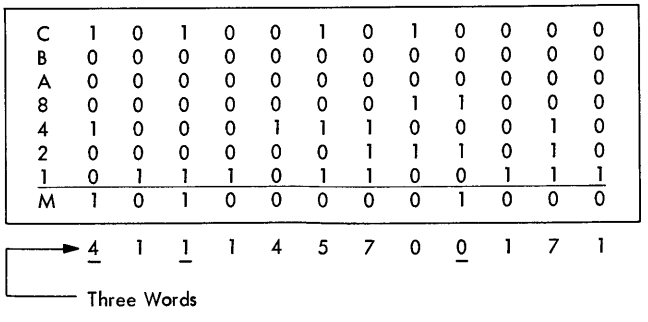


Figure 7. Bit Configuration of Three Words in Storage

CHARACTER	BINARY CODE			CARD CODE
	C	BA	8421	
0	1	00	1010	0
1	0	00	0001	1
2	0	00	0010	2
3	1	00	0011	3
4	0	00	0100	4
5	1	00	0101	5
6	1	00	0110	6
7	0	00	0111	7
8	0	00	1000	8
9	1	00	1001	9
A	0	11	0001	12-1
B	0	11	0010	12-2
C	1	11	0011	12-3
D	0	11	0100	12-4
E	1	11	0101	12-5
F	1	11	0110	12-6
G	0	11	0111	12-7
H	0	11	1000	12-8
I	1	11	1001	12-9
J	1	10	0001	11-1
K	1	10	0010	11-2
L	0	10	0011	11-3
M	1	10	0100	11-4
N	0	10	0101	11-5
O	0	10	0110	11-6
P	1	10	0111	11-7
Q	1	10	1000	11-8
R	0	10	1001	11-9
S	1	01	0010	0-2
T	0	01	0011	0-3
U	1	01	0100	0-4
V	0	01	0101	0-5
W	0	01	0110	0-6
X	1	01	0111	0-7
Y	1	01	1000	0-8
Z	0	01	1001	0-9
&	1	11	0000	12
.	0	11	1011	12-3-8
□	1	11	1100	12-4-8
-	0	10	0000	11
$	1	10	1011	11-3-8
*	0	10	1100	11-4-8
/	1	01	0001	0-1
,	1	01	1011	0-3-8
%	0	01	1100	0-4-8
#	0	00	1011	3-8
@	1	00	1100	4-8
blank	1	00	0000	

Figure 8. Binary Coding of Characters in an IBM 1401 System

Unless you have had previous computer experience, an electronic internal storage device is a pretty awesome thing to contemplate, especially when some people call it *memory* and others refer to computers as *giant brains.* External storage devices we can see. Internal storage devices are hidden behind the covers of the computer. Even when you remove the covers and take a good look, your reaction is apt to be one of bewilderment.

Memory and *brains* imply the ability to think and reason, and make decisions based on an analysis of facts or assumptions. Do computers really *think?*

There are two schools of thought among scientists and electronic engineers. One group says computers think. The other says computers are morons that can do an awful lot of work during the flick of an eyelash, but need someone with a brain to tell them what to do.

Like the human brain, an internal storage unit is able to accept, remember, and send out information given to it. Unlike the human brain, however, it is not capable of creative thought.

A computer can read a record, process it, and write a line. It can tell the difference between one record and another. It can position paper in the printer at the right place and at the right time. It can accumulate and print totals anywhere you want them on paper, and when you want them. It can add, subtract, multiply, divide by repetitive additions or subtractions. It can do square roots and cube roots of any number you choose. The truth of the matter is that a computer won't do any one of these things (not even feed a card from the hopper) without someone to tell it, by means of a stored program, when and how to do it.

One other point is worth mentioning. You may know everything there is to know about a computer from A to Z, but if you don't know how to go about solving problems without a computer, you can't expect a computer to do it for you. We repeat, *a computer is not capable of creative thought.*

You could tell a computer to multiply one figure by another and get the wrong answer if your MULTIPLY instructions are not specific to the last detail. Computers give you the right answers when the program is right, but they can also give you the wrong answers (incidentally, just as fast) when the program has bugs.

Programming and coding involve so much detailed work that most programs do not operate correctly when first tried. Some programs never leave home plate, while others keep circling the bases over and over in a never-ending loop until someone stops the machine.

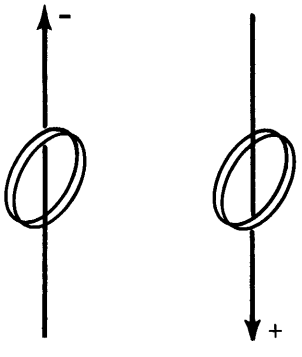


Figure 9.

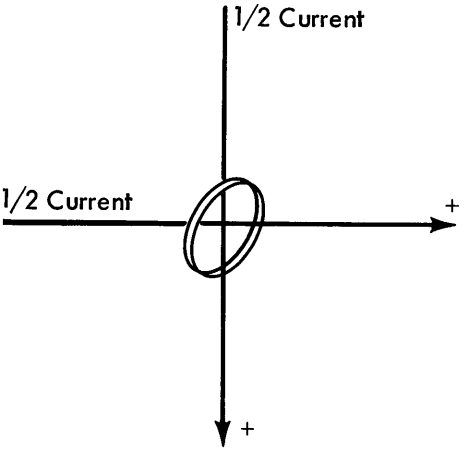


Figure 10.

Thus, it is necessary to *debug* the program (locate and correct the errors), and to test it with test cases to be sure that it properly processes.

All of this goes under the name of *program check-out.* And to locate errors in your program, you've got to know something about the internal storage unit, and how you can track down the errors. There are several types of internal storage units. The type used in the 1401 is known as *magnetic core storage.*

Magnetic Core Storage

A magnetic core-storage unit is an extremely high-speed device, able to *read in* and *read out* data in a few millionths of a second.

A magnetic core is a tiny, doughnut-shaped object made of a ferromagnetic material. It can easily be magnetized, and unless deliberately changed will retain its magnetism indefinitely.

If we pass a wire through the center of this core, and send a sufficiently strong current through it, the core becomes magnetized. The direction of the current determines the polarity of the magnetic field. We speak of the core as being positively or negatively magnetized (Figure 9). A positive core has a value of 1, a negative core has a value of 0.

By passing two wires through the core, and sending just half the amount of current necessary to magnetize the core through each, we make sure that the core will become magnetized only when both wires carry current (Figure 10).

Now, we can string many cores on a screen of wires (but still only magnetize one of the cores) by sending current through the appropriate pair of wires. In Figure 11, we magnetized the shaded core by sending current through one horizontal and one vertical wire. No other core is magnetized, because only the desired core receives enough current to magnetize it. By passing current through other pairs of wires, we can control the magnetism of any of the cores.

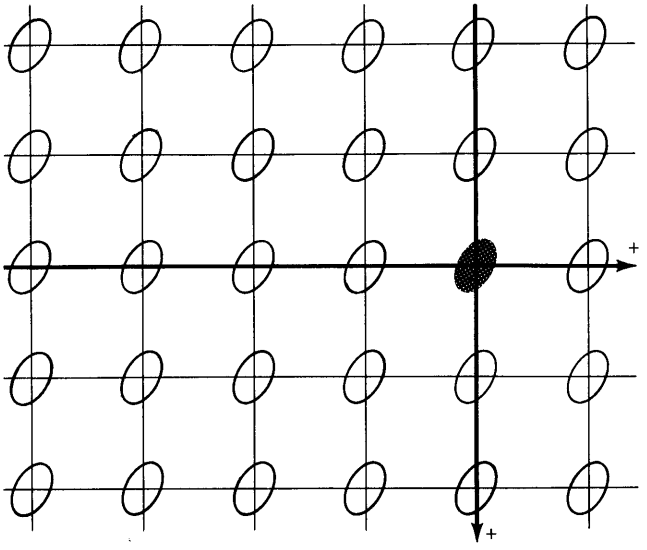


Figure 11.

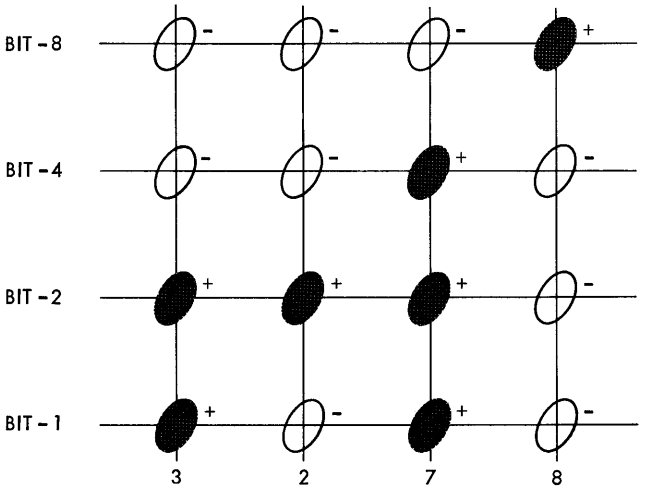


Figure 12.

We can control the cores in two ways. Through selection of a specific pair of wires, we can decide which core is to be affected. By controlling the direction of current flow, we determine whether it is to be positively or negatively magnetized.

Let's see how we can store a four-digit number in magnetic core memory, using the binary system. In the binary system we use combinations of the numbers 1, 2, 4, and 8 to express any digit. So we need four rows, or bits, of cores. We also need four columns of cores to accommodate the four digits of our example (say 3278). In the units position, we need to positively magnetize only one core in bit position 8 (Figure 12) to represent the 8 of our number. In the tens column we must positively magnetize three cores (in bit positions 1, 2, and 4) that add up to the 7.

Using the same principle, we read the other digits into storage. All other cores are magnetized negatively to indicate that they are of zero value. In actual practice, more bits, or rows of cores are used to provide for alphabetic storage, plus-or-minus indication, and checking.

To read information from magnetic core memory, we must use a third wire (called *sense wire*) that passes through the cores. Information is read by changing a core from positive to negative. The change causes a small current in the sense wire. The current can be read and amplified for use in the machine circuit. If the core was negative to begin with, no current flow is induced in the sense wire. However, changing the cores from positive to negative also destroys the information stored. To retain a given bit of information, therefore, we must cause a *read in* to take place following each *read out.*

Organization of Core Storage

Core storage units in the 1401 come in six sizes. These sizes (1400, 2000, 4000, 8000, 12000, and 16000) indicate the number of characters each size accommodates.

CODED ADDRESSES IN STORAGE		
ACTUAL ADDRESSES		3-CHARACTER ADDRESSES
000 to 999	No zone bits	000 to 999
1000 to 1099		⊕00 to ⊕99
1100 to 1199		/00 to /99
1200 to 1299		S00 to S99
1300 to 1399		T00 to T99
1400 to 1499	A-bit,	U00 to U99
1500 to 1599	using 0-zone	V00 to V99
1600 to 1699		W00 to W99
1700 to 1799		X00 to X99
1800 to 1899		Y00 to Y99
1900 to 1999		Z00 to Z99
2000 to 2099		I 00 to I 99
2100 to 2199		J00 to J99
2200 to 2299		K00 to K99
2300 to 2399		L00 to L99
2400 to 2499	B-bit,	M00 to M99
2500 to 2599	using 11-zone	N00 to N99
2600 to 2699		*O00 to O99
2700 to 2799		P00 to P99
2800 to 2899		Q00 to Q99
2900 to 2999		R00 to R99
3000 to 3099		?00 to ?99
3100 to 3199		A00 to A99
3200 to 3299		B00 to B99
3300 to 3399		C00 to C99
3400 to 3499	A-B-bit,	D00 to D99
3500 to 3599	using 12-zone	E00 to E99
3600 to 3699		F00 to F99
3700 to 3799		G00 to G99
3800 to 3899		H00 to H99
3900 to 3999		I00 to I99

* Letter O followed by Zero Zero

Figure 13. Core-Storage Address Codes

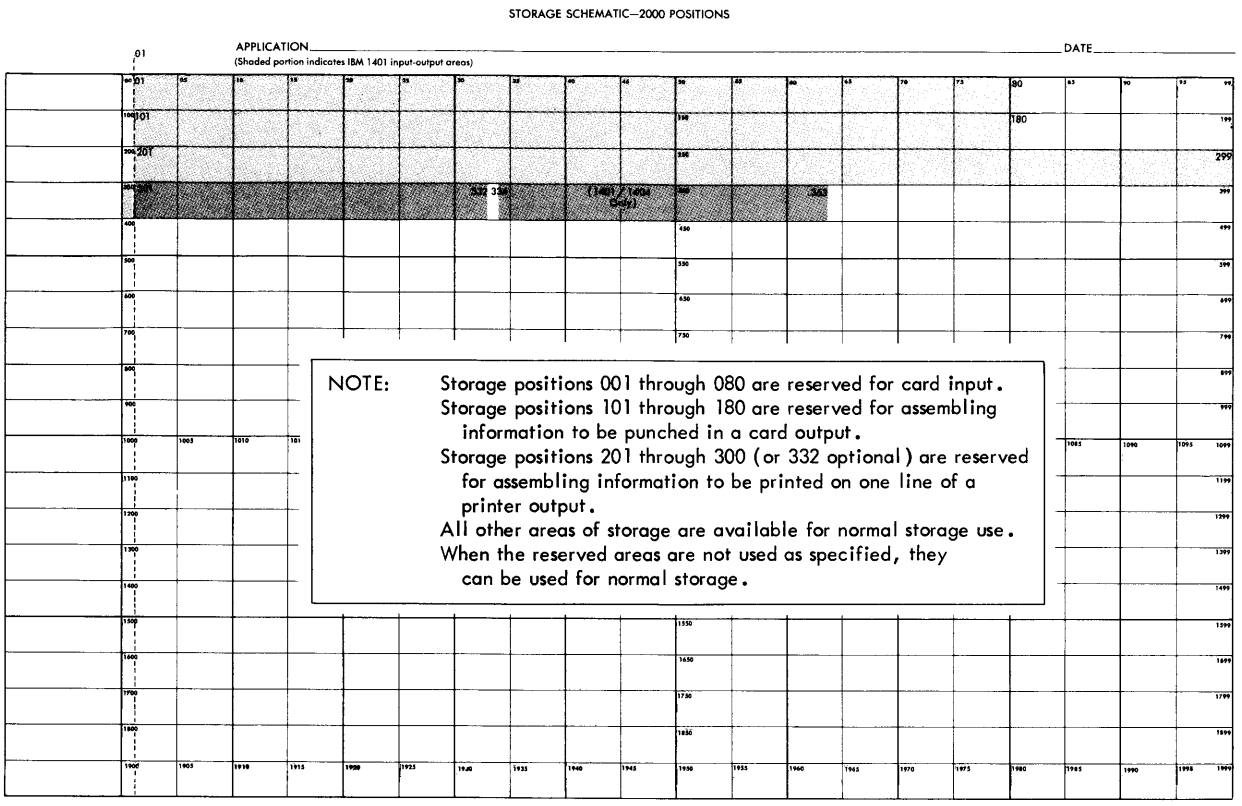


Figure 14. Core-Storage Layout Chart

Programmers refer to these sizes as 1.4k, 2k, 4k, 8k, 12k, and 16k (k=1000 characters).

Each character position is numbered, starting with zero. Thus, the positions in a 1.4k storage unit are numbered 0000 to 1399, in a 2k 0000 to 1999, in a 4k 0000 to 3999, etc. These numbers are known as *addresses*. They play a major role in:

1. locating the program where you want it in storage.
2. executing the instructions in the program sequentially, or in some prescribed sequence.

An instruction always specifies an operation to be performed. But in many cases, it must also indicate:

A. Where the data that is to be operated on is located in storage.

B. Where to place the results.

It's like an envelope with two addresses: one address (A) tells where the letter is coming from, and the other address (B) tells where it is going.

When these address locations are used as part of an instruction, they are referred to as *A- and B-addresses, A- and B-operands,* or *A- and B-fields.* They occupy a major portion of the storage area required to store the program.

To conserve storage as much as possible, the engineers developed a coding system so that addresses 0000-3999 can be referenced by using three characters instead of four. This is done by using digits 0-9, letters A-Z, and four special characters ≠ / ! and ? as the first position of the three-character address and 00-99 as the other two. The numbers, letters and special characters were assigned in the sequence shown in Figure 13, and laid out as shown in Figure 14.

Addresses 0001-0999 are easy to remember. You simply drop the high-order digit, and use 001-999 instead. Position 0000 is not addressable. For addresses 1000 to 3999, you better use the code chart. In Figure 14, note the areas reserved for card input, card output, and printer output.

For the larger storage units (8k, 12k, and 16k), a 0-zone punch (A-bit) in the units position of an address represents the 4000 block of numbers from 4000-7999. The 11-zone punch (B-bit) represents the next 4000 block of numbers from 8000-11999. The 12-zone punch (AB-bits) represents the last 4000 block of numbers from 12000-15999. Thus, the three-character address for 4323 is 32T. The T in the units position means 4--3, and the other two digits in between are supplied by 32.

The three-character address for 15323 is C2C. The C in the units position stands for 12--3, the C in the hundreds position for -33--, and the 2 in the tens position for 2. The computer assembles these combinations together and comes up with 15323.

12--3 (C in units position)
-33-- (C in hundreds position)
---2- (2 in tens position)

15323

It is obviously impractical to try to memorize combinations like these. Use the code chart in Figure 13 for addresses 1000-3999. For addresses 4000-15999, use the condensed code chart shown in Figure 15.

Instruction Word Format

A 1401 instruction in actual machine language may vary from one to eight characters. The first character of an instruction is always an operation code with a word mark. Figure 16 shows a complete list of 1401 operation codes.

Some instructions require no operands. These are usually input-output instructions such as READ A CARD (1), WRITE A LINE (2), or PUNCH A CARD (4).

Other instructions, such as a branch to a specific location in storage (say B892) require an A-field. This is an *unconditional branch*, and in effect says to the computer *When you get this far, always branch to location 892 where you'll find an instruction that will tell you what to do next.*

Most instructions are followed by two operands. *Example:* A534655. This instruction says to the computer *Add what you have in the A-field to what you have in the B-field.* The number 534 is the location of the low-order position of the A-field and the number 655 is the location of the low-order position of the B-field.

There are still other types of instructions that are modified by a d-character. In effect, this type of instruction says *Do this and that IF*. Depending on the instruction, the d-character may be the second, fifth or eight position of the instruction.

Figure 17 shows nine examples of 1401 instructions that vary from one to eight characters.

Problem Solving

To solve a problem, we must first define the problem. By defining a problem, we mean: having a clear picture of the results we want, breaking the problem down into its component parts, and then developing a logical procedure to put the parts together.

It's like working out a jigsaw puzzle. The problem is partially defined for you with an assembled picture that usually comes with the set. Putting the pieces together to make the picture come out like the sample is rarely accomplished by trial and error. You develop a plan, and proceed according to the plan. If there are only two pieces, a child can put them together. Some jigsaw puzzles developed for children often involve as many as 15 or 20 pieces. But jigsaw puzzles of several hundred or several thousand pieces are not for children. They are for grownups. So, what do you do to put the puzzle together? Where do you start?

Well, first you study the sample picture and get some idea of its size, coloring, and picture content. The picture may be a landscape, with a *red* barn in the center, a winding dirt road (*brown*) leading up to it, and some ever*green* trees here and there. A few farm animals are grazing in a meadow (*light green*) on the left side of the road. The sky is *blue*, sprinkled with large *white* clouds.

You pick up a couple of pieces that look as if they might fit together. They don't. You try a couple more with the same results. With several hundred pieces to put together, the task seems hopeless.

ACTUAL ADDRESSES	ZONE BITS OVER HUNDREDS POSITION	ZONE BITS OVER UNITS POSITION	3-CHARACTER ADDRESSES
0000 to 0999	No Zone Bits	No Zone Bits	000 to 999
1000 to 1999	A-Bit (Zero-Zone)	No Zone Bits	≠00 to Z99
2000 to 2999	B-Bit (11-Zone)	No Zone Bits	!00 to R99
3000 to 3999	AB-Bits (12-Zone)	No Zone Bits	?00 to I99
4000 to 4999	No Zone Bits	A-Bit (Zero-Zone)	00≠ to 99Z
5000 to 5999	A-Bit (Zero-Zone)	A-Bit (Zero-Zone)	≠0≠ to Z9Z
6000 to 6999	B-Bit (11-Zone)	A-Bit (Zero-Zone)	!0≠ to R9Z
7000 to 7999	AB-Bits (12-Zone)	A-Bit (Zero-Zone)	?0≠ to I9Z
8000 to 8999	No Zone Bits	B-Bit (11-Zone)	00! to 99R
9000 to 9999	A-Bit (Zero-Zone)	B-Bit (11-Zone)	≠0! to Z9R
10000 to 10999	B-Bit (11-Zone)	B-Bit (11-Zone)	!0! to R9R
11000 to 11999	AB-Bits (12-Zone)	B-Bit (11-Zone)	?0! to I9R
12000 to 12999	No Zone Bits	AB-Bits (12-Zone)	00? to 99I
13000 to 13999	A-Bit (Zero-Zone)	AB-Bits (12-Zone)	≠0? to Z9I
14000 to 14999	B-Bit (11-Zone)	AB-Bits (12-Zone)	!0? to R9I
15000 to 15999	AB-Bits (12-Zone)	AB-Bits (12-Zone)	?0? to I9I

Figure 15. Increased Core-Storage Address Codes

Figure 16. IBM 1401 Operation Codes

INPUT-OUTPUT CODES

OPERATION CODE	FUNCTION	MNEMONIC	BCD CODE	CARD CODE
1	Read a Card	R	1	1
2	Write a Line	W	2	2
2 □	Write Word Marks		□ is modifier	
3	Write-Read	WR	C21	3
4	Punch a Card	P	4	4
4R	Read-Punch Feed*		R is modifier	
4(I)R	Read-Punch Feed and Branch*		R is modifier	
5	Read-Punch	RP	C41	5
6	Write-Punch	WP	C42	6
6R	Write-Read Punch Feed*		R is modifier	
6(I)R	Write-Read Punch Feed and Branch*		R is modifier	
7	Write-Read-Punch	WRP	421	7
8	Start Read Feed*	SRF	8	8
9	Start Punch Feed*	SPF	C81	9

ARITHMETIC CODES

A	Add	A	BA1	12-1
S	Subtract	S	CA2	0-2
?	Zero and Add	ZA	CBA82	12-0
!	Zero and Subtract	ZS	B82	11-0
@	Multiply*	M	C84	4-8
%	Divide*	D	A84	0-4-8

LOGIC OPERATION CODES

B(I)	Branch	B	BA2	12-2
B(I)d	Branch if Indicator ON		d is modifier	
B(I)(B)d	Branch if Character is Equal	Contents of B compared to d		
V(I)(B)d	Branch if WM and/or Zone	BWZ	A41	0-5

MOVE AND LOAD CODES

D	Move Numerical	MN	BA4	12-4
L	Load Character to A Word Mark	LCA	B21	11-3
M	Move Characters to A or B Word Mark	MCW	CB4	11-4
Y	Move Zone	MZ	CA8	0-8
Z	Move Characters and Suppress Zeros	MCS	A81	0-9
,	Set Word Mark	SW	CA821	0-3-8
□	Clear Word Mark	CW	CBA84	12-4-8

* Special Feature

MISCELLANEOUS OPERATION CODES

OPERATION CODE	FUNCTION	MNEMONIC	BCD CODE	CARD CODE
C	Compare	C	CBA21	12-3
E	Move Characters and Edit	MCE	CBA41	12-5
F	Control Carriage	CC	CBA42	12-6
H	Store B-Address Register*	SBR	BA8	12-8
K	Select Stacker	SS	CB2	11-2
N	No Operation	NOP	B41	11-5
Q	Store A-Address Register*	SAR	CB8	11-8
/	Clear Storage	CS	CA1	0-1
.	Halt	H	BA821	12-3-8
#	Modify Address*	MA	821	3-8

CHARACTER AT d FOR B(I)d BRANCH

d	BRANCH ON	d	BRANCH ON
b1	Unconditional	R	Carriage Busy*
9	Carr. Chan. #9	T	Low Compare B < A*
A	"Last Card" Switch	U	High Compare B > A*
B	Sense Switch B*	Z	Overflow
C	Sense Switch C*	?	Reader Error if I/O Check Stop Switch OFF
D	Sense Switch D*	!	Punch Error if I/O Check Stop Switch OFF
E	Sense Switch E*	≠	Printer Error if I/O Check Stop Switch OFF
F	Sense Switch F*	@	Carr. Chan. #12
G	Sense Switch G*	%	Processing Check with Process Check Switch OFF
K	End of Reel*	/	Unequal Compare B ≠ A
L	Tape Error*		
S	Equal Compare B = A*		
P	Printer Busy*		

CHARACTER AT d FOR MAGNETIC TAPE / DISK STORAGE

d	OPERATION	d	BRANCH ON
B	Backspace Tape Record	N	Access Inoperable
E	Skip and Blank Tape	V	Read/Write Parity Check or Read Back Check Error
M	Write Tape Mark	W	Wrong-Length Record
R	Rewind Tape	X	Unequal Address Compare
U	Rewind Tape and Unload	Y	Any Disk Storage Error Condition

CHARACTER AT d FOR 1407 CONSOLE INQUIRY STATION

Q	Inquiry Request	*	Inquiry Clear

MAGNETIC TAPE %UX TAPE UNIT ADDRESS

INSTRUCTION	FUNCTION	MNEMONIC	BCD CODE	CARD CODE
L(%UX)(B)d	Read/Write Tape with Word Marks	LCA	d-modifier, R-Read Tape W-Write Tape	
M(%UX)(B)d	Read/Write Tape	MCW		
M(%CX)(B)R	Read Compressed Tape*	(%CX) is address of tape unit		
P(A)(B)	Move Characters to Record or Group Mark*	MCM	CB421	11-7
U(%UX)d	Control Unit	CU	CA4	0-4
X(A)(B)	Move and Insert Zeros*	MIZ	CA421	0-7

INSTRUCTION	FUNCTION	REMARKS
COLUMN BINARY		
1C	Read Column Binary	C is Modifier
4C	Punch Column Binary	C is Modifier
M(A)(B)A	Move and Binary Decode	A is Modifier
M(A)(B)B	Move Binary Code	B is Modifier
M(%BX)(A)R	Read Binary Tape	%BX is Address of tape unit
M(%BX)(A)W	Write Binary Tape	
W(I)(B)d	Branch if Bit Equal	BBE is mnemonic
DISK STORAGE %FX DISK OPERATION		
M(%F0)(B)R	Seek Disk	B is Disk Address
M(%FX)(B)R	Read Disk	X can be 1, 2, or 3 1 Specifies Single Record 2 Specifies Full Track 3 Specifies a Write Disk Check operation M(%F3)(B)W
M(%FX)(B)W	Write Disk	
L(%FX)(B)R	Read Disk with Word Marks	
L(%FX)(B)W	Write Disk with Word Marks	
1407 INQUIRY %TO ADDRESS		
M(%T0)(B)R	Read Console Printer	Data from 1407 transferred to B-address
M(%T0)(B)W	Write Console Printer	Data at B-address transferred to 1407
L(%T0)(B)R	Read Console Printer with Word Marks	Data from 1407 transferred to B-address with Word Marks
L(%T0)(B)W	Write Console Printer with Word Marks	Data at B-address transferred to 1407 with Word Marks
M(%T0)(B)W	Line Space	B is address of a Group Mark with a Word Mark

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION

IBM 1401 AND 1440 PROGRAM CHART

FORM X24-6437
PRINTED IN U.S.A.

Program:_____

Programmer:_____ Date:_____

Step No.	Inst. Address	Instruction OP	A/I	B		Remarks	Effective No. of Characters Inst.	Data	Total
	5–7	12			19	←(card columns in instruction cards)			
XX	XXX	I				Read a Card	1		
XX	XXX	K	2			Select a card in Stacker #2	2		
XX	XXX	B	400			Unconditional branch to high-order position of instruction word in Location 400	4		
XX	XXX	2	650			Print a Line and unconditional branch to high-order position of instruction word in Location 650	4		
XX	XXX	3	003	067		Set a word mark in Locations 003 and 067 (high-order positions of data words)	7		
21	502	C	008	758		Compare word in Location 008 with word in Location 758 (low-order positions of words)	7		
22	509	B	625	1		Branch to high-order position of instruction word in Location 625 IF compare operation was unequal	5		
XX	XXX	A	072	423		Add word in Location 072 to word in Location 423 (low-order positions of words). In this case, B-operand 423 is the low-order position of an area in storage to be used as an "accumulator."	7		
XX	XXX	B	650	080	–	Branch to high-order position of instruction word in Location 650 IF Location 080 has a B-bit (X punch in 80)	8		
						NOTE: Word marks in high-order positions of above instructions are automatically set during the Program Loading Routine. The instructions are not presented as a complete program for a problem. They merely show examples of variable-length instructions. A 3-character address is required in the column headed "Inst. Address" to indicate the exact location of the high-order position of each instruction word, as shown in the two sequential instructions (Steps 21 and 22).			
						The instruction address is punched in columns 5-7 of the instruction cards and the instruction is punched in columns 12-19.			

Figure 17. Instruction Word Formats

Then you start to think. All of a sudden you realize that the pieces that make up the outer edges of the picture must obviously have straight edges. The four pieces having two straight edges at right angles, if you can find them, must be the corner pieces. So, with the four corner pieces and the pieces with straight edges, you build a frame. By observing the color of each piece and matching the color with the colors on the sample picture, you begin to fit the pieces into place one at a time. A little trial here and there, and a little luck now and then, and before long you have all the pieces in the right places, and your puzzle is solved. You pat yourself on the back, and with confidence that jigsaw puzzles are not so tough after all, you tackle another one. And the more of them you do, the faster you can put them together.

This, of course, is essentially programming – programming not from any set of written instructions, but from memory, use of logic, and common sense. If you go to the trouble of numbering each piece as it fits into place, you can solve the same puzzle next time without any thinking, but you won't have as much fun. You simply fit the pieces together with a number system in a fraction of the time it took you the first time you solved the problem. And you can do this even without a picture to guide you.

Believe it or not, a program for a computer is developed in much the same way. Knowing what you want to accomplish, you design a printer form with the aid of the printer spacing chart. This is the picture you want. You then study the information in the basic records, find out where each item is located in the basic records, and compare it with the information you want (and where you want it) in your results. These are the pieces with the straight edges. You then plan the procedure you are going to follow with a block diagram. This is the frame. In developing the block diagram, you look for the names of the items in your basic records, and associate them with similar names in the picture you specified at the outset. These are the colors.

From that point on, you apply your knowledge of the computer, and develop the program step by step. As each step is taken, you give it a number, write the instruction in machine or symbolic language, and proceed with the next step. Because the program is to be executed for each record, the last instruction in your program must *branch* back to the start of the program, which may or may not be the first instruction written.

Like a child who has no trouble putting two pieces of a jigsaw puzzle together, no one has any trouble understanding one isolated 1401 instruction or two sequential 1401 instructions. It's the positioning of each instruction in the body of the program that requires brain power – brain power that the computer does not have.

In order to learn 1401 programming step by step, we shall feel our way with a few simple problems in the beginning. Assume that you are a new employee in a data processing installation, and it's your job to program the problems as they come up. The problems progress from the simple to the complex.

Each new problem requires one or more operations not used in an earlier problem. In that way, we can readily profit from the experience gained in solving one problem by applying it in solving a later problem. No matter how simple, problems are defined, and specifications set up in a block diagram. A solution will be developed with instructions written on an *IBM 1401 Program Chart, Form X24-6437.*

The XYZ Company

The XYZ Company is a mythical company that manufactures ten models of radio sets and five models of television sets. Its home office and manufacturing plant are in a large eastern city. It employs over 3,000 people at the main office and plant, and over 200 salesmen who work out of branch offices scattered over some 30 cities, most of them east of the Mississippi River. About 150 of the employees at the plant are scientists and electronic engineers, a few of whom are engaged in electronic research. It has great growth potential.

Ninety percent of its products are sold directly to over 2,000 local dealers. The other ten percent are sold to three large mail order houses.

The data processing department (the 1401 and several pieces of unit record equipment) controls about 95% of the operations in the field and at the home office. The jobs now being done on this equipment are as follows:

Weekly payroll registers
Weekly checks and earnings statements to employees
Weekly job time distribution to various manufacturing accounts
Daily production control reports
Weekly manufacturing cost reports
Weekly stock reports and inventory reports
Daily billing
Monthly sales reports by branch offices and salesmen
Semimonthly commission statements to salesmen
Accounts receivables
Monthly statements to dealers and mail order houses
Accounts payables
Weekly remittance statements and voucher checks to vendors
General accounting
Monthly income and expense reports for management
Quarterly financial statements for management and stockholders
Quarterly social security reports and income tax reports to the U. S. Internal Revenue Service
Annual W-2 forms for U. S. Treasury and employees
Annual financial reports for management and stockholders

At the present time, all of the records involved in processing these reports are in punched card files, but the company plans to convert some of them to magnetic tapes within the next year. Plans beyond that period include RAMAC® disk files, which a team of systems planners and systems analysts are now considering.

You have the responsibility of handling the card files now involved in preparing the weekly payroll register, and the weekly checks and employees earnings statements. Figures 5 and 6 show the basic record and card form used in this application. Other card forms will be presented as the need arises.

Although programs have been developed for your part of the work, the supervisor has given you permission to write your own programs. There is enough 1401 time available to permit you to do a little experimenting now and then. Besides, you need programming experience, and your supervisor is willing to give you the chance, when it does not interfere with the present schedules. You're on your own for the most part, but help is available from several experienced

programmers in the department, including the supervisor, whenever you need it.

Problem 1. Reproducing

One of the files maintained by the XYZ Company is an *Employee Payroll Master Name Card* file. The file consists of one card (Figure 18) for each employee on the payroll. The card fields from left to right read as follows:

Fields	*Columns*
Man No.	3-8
Dept. No.	9-12
Marital Status	49
Tax Code	50
First Initial	51
Second Initial	52
Name	53-70
Social Security Number	71-79
Card Code (X)	80

This file is used primarily in conjunction with other types of cards to prepare the weekly payroll register, and the payroll check and earnings statement. It is also used periodically to prepare registers of employees in alphabetical order, in man number order, and in social security number order, as well as tabulations by marital status and tax code.

The last time the file was used, you had several card feed failures. It's time to reproduce a new deck. Ordinarily, a simple reproducing job of this type would be done with a reproducer. There is one in the installation, but it's going to be tied up for several days on a more urgent job.

You are fully aware that to tie up a whole system as powerful as the 1401 (Figure 19) to do a simple odd job that can be done with other unit record equipment is like using a 10-ton truck to pick up a 3-pound package at the grocery store three blocks away. But the 1401 is unscheduled for the next several hours, and you decide to reproduce the file using the 1401.

For the reproducing job you are about to do, you need only the IBM 1402 Card Read-Punch and the IBM 1401 Processing Unit. For a general description of these units, refer to the 1401 reference manual.

In the processing unit, you will need the area that is reserved for card input (storage locations 001-080), and the area that is reserved for card output (storage locations 101-180). Another area will be needed, of course, to store the program.

Because the individual fields in the employee payroll master name card need not be defined, you need to set a word mark only in location 001. The instructions you will need, then, are:

Set a word mark
Read a card
Move or load a record from the card read area (001-080) to the card punch area (101-180)
Punch a card
Branch to read a card

After looking up these instructions in the 1401 reference manual for detail descriptions, you are ready to draw a block diagram.

Block Diagram

A block diagram is a graphic presentation of the steps in a program. Its primary purpose is to establish the sequence in which the operations will be performed, and to serve as a basis for coding. The amount of detail included in a block diagram varies. In some cases, where the details can be assumed, the block diagram can be quite general in nature. In other cases, it may be necessary to include the details in order to make it easy to prepare the program coding sheet.

The first block diagram you draw for *Problem 1* is shown in A of Figure 20. The diagram consists of five blocks. Each block specifies an operation in general terms. After studying this block diagram, you decide to take advantage of a shortcut, and you come up with the block diagram in B of Figure 20. The program in B will reproduce your master file faster than the program in A, because in step 4 the card read operation is combined with the card punch operation. This results in a maximum production of 250 cards a minute. For technical description of *timing*, see *Card Reader, Card Punch*, and *Combination Read and Punch* of the *Timing Input and Output Operations* section of the 1401 reference manual.

Writing the Program in Actual Machine Language

The program you are about to write in actual machine language can be loaded in any area of storage you specify that is not required for input and output storage. You decide to load the first instruction in location 400. You write this number in the column headed *Inst. Address* of the 1401 program chart, as shown in Figure 21.

The first instruction (,) is a SET WORD MARK instruction with an A-operand (001). This instruction will occupy four positions of core storage (400-403), and will set a word mark in location 001.

The second instruction (1) will read a card, and will occupy one position of core storage (404).

The third instruction (L080180) is a load instruction with an A-operand (080) and a B-operand (180). It will occupy storage positions 405-411. This instruction will move the data in the read area to the punch area,

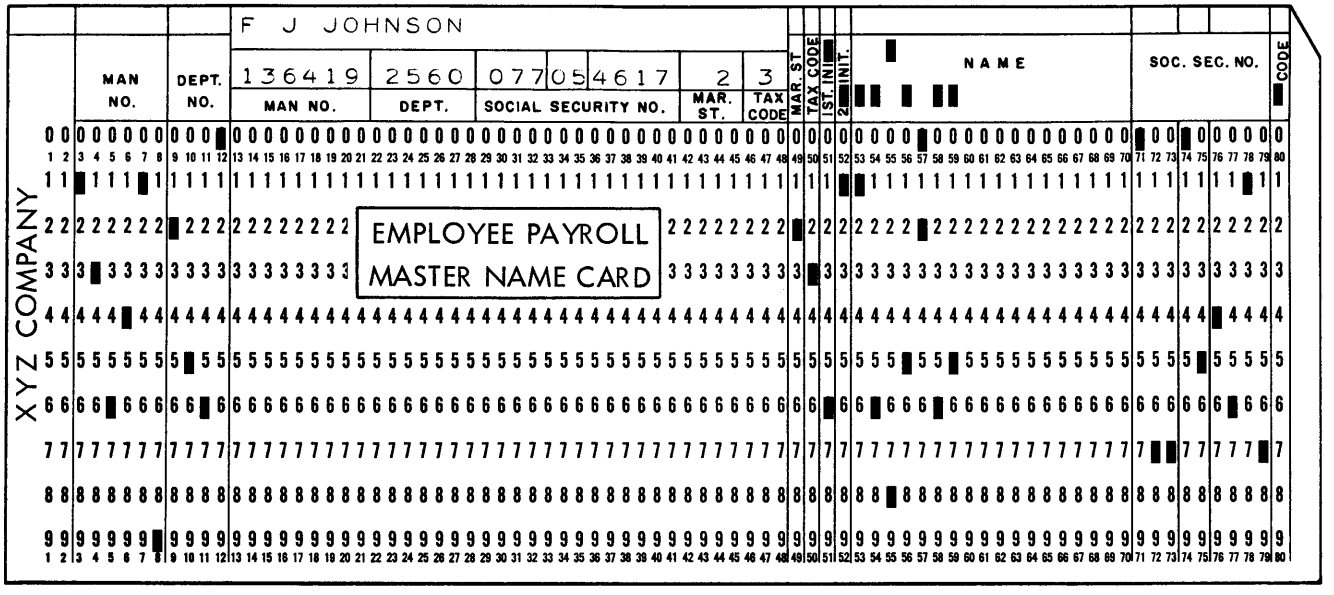
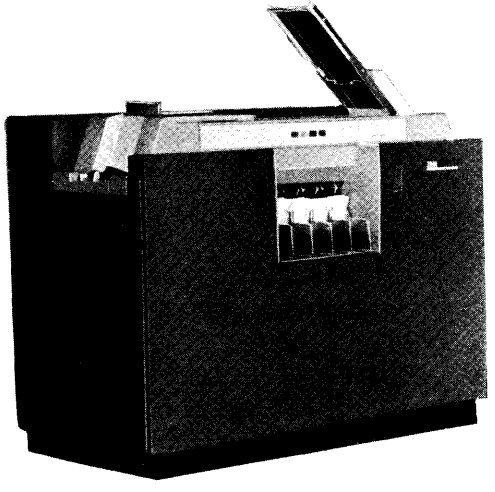
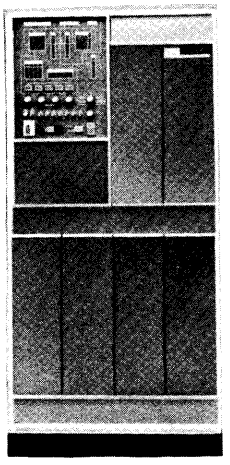


Figure 18. Employee Payroll Master Name Card



IBM 1402 Card Read-Punch



IBM 1401 Processing Unit
(1400 core-storage positions)

IBM 1403 Printer

Figure 19. IBM 1401 Data Processing System

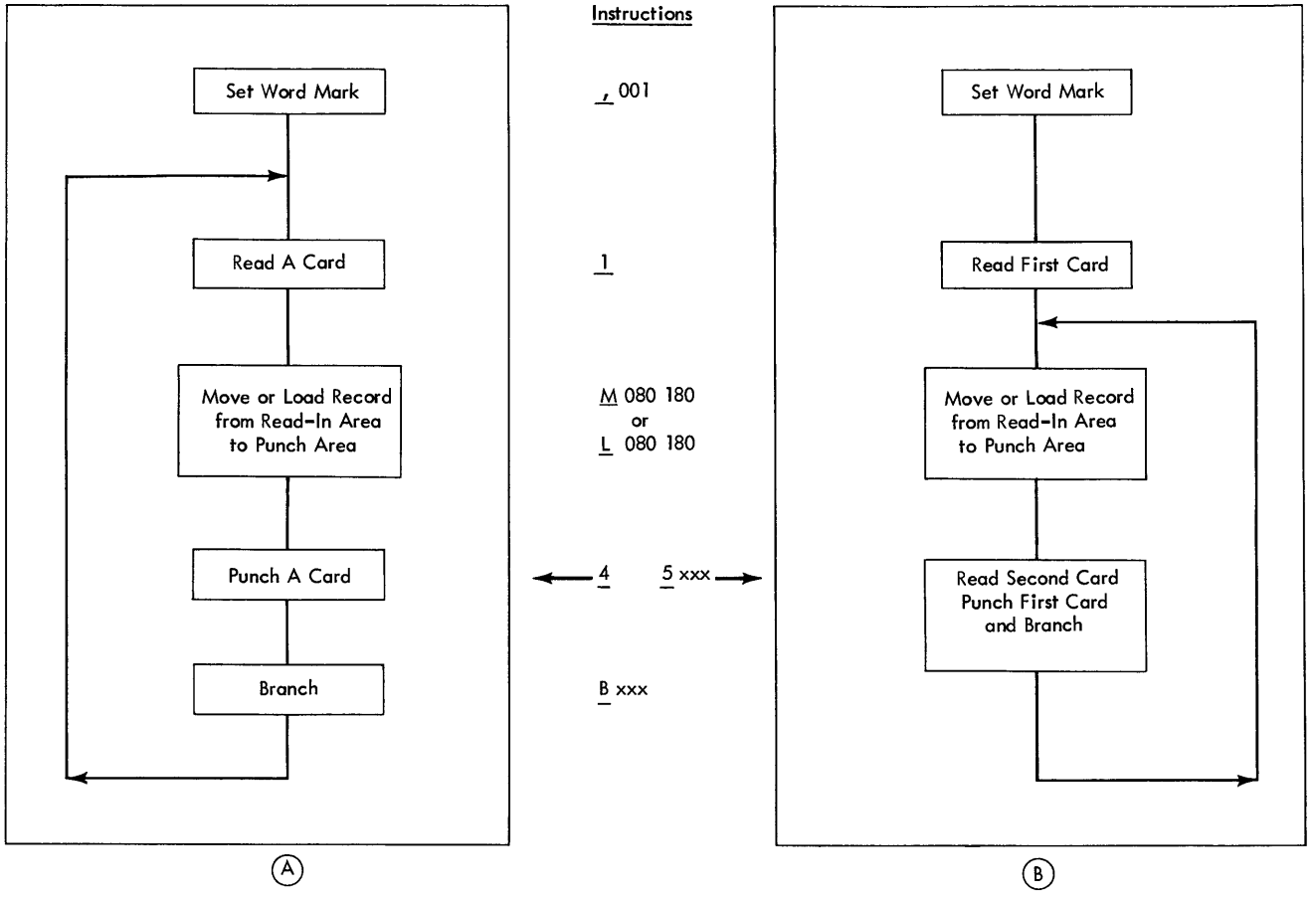


Figure 20. Block Diagrams for Problem 1

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION

IBM 1401 AND 1440 PROGRAM CHART

FORM X24-6437
PRINTED IN U.S.A.

Program: Reproduce Employee Payroll Master Name Card File

Programmer: _____ Date: _____

Step No.	Inst. Address	Instruction OP	A/I	B	Remarks	Effective No. of Characters: Inst.	Data	Total
	5-9	12		19	←---(card columns in instruction cards)			
1	400	9	001		Set Word Mark in 001	4		4
2	404	1			Read a Card	1		5
3	405	L	080	180	Load card data from read-in area to punch area	7		12
4	412	5	405		Read, Punch and Branch to Location 405	4		16
5	416				Set Word Mark at end of program	1		17
					NOTE: Clear and branch card is punched /400080 in columns 60-66.	17		

Figure 21. Program Chart for Problem 1

column for column. The word mark in 001 (set by the first instruction) will also be moved to location 101, but any word marks in storage from 102 to 180 will be erased.

The fourth instruction (5) is a combination instruction to read one card and punch another simultaneously. It will occupy location 412. The A-operand to this instruction (405) will occupy locations 413-415, and will cause a branch to location 405 to repeat the operation for the next card.

To signal the end of the program, insert an instruction card in step 5, as shown in Figure 21. This card will place a word mark in location 416. This is the simplest way to signal the end of a program.

Punching the Instruction Cards

Turn your 1401 program chart over to a key punch operator who is trained to punch instruction cards. Although your program consists of only five steps, and therefore five instruction cards, they must be preceded by three leader cards and followed by one CLEAR AND BRANCH CARD.

The procedure is described in *Rule 1* of the *Program Loading Routine* section of the 1401 reference manual. Note carefully the card formats for the leader cards, instruction cards, constant cards (none are required in your program), and the CLEAR AND BRANCH card.

In this case, the CLEAR AND BRANCH instruction is /400080. The slash (/) is the operation code for CLEAR. The A-operand (400) is the location of the first instruction in your program you want to branch to. The B-operand (080) is the low-order position of the card input area you want to clear.

Figure 22 shows the nine self-loading instruction cards for *Problem 1.*

Loading and Testing the Program

Before you start the job, refer to the section on *IBM 1401 Console Keys, Lights, and Switches* in the 1401 reference manual. This section, supplemented by a figure of the 1401 console, tells you when and how to use the various keys, switches, dials, and lights. For your job, it is only necessary to set the mode switch on RUN and to press the check reset key and the start reset button.

Next refer to the *IBM 1402 Card Read-Punch* section of the reference manual. This describes generally the card read-punch unit. Then see the section in the reference manual on *IBM 1402 Card Read-Punch Operating Keys, Lights and Switches.* This describes the purpose of each key, switch, and light on the card read-punch.

Now place blank employee payroll master name cards in the punch hopper, 12-edge first, face down. Then place the nine instruction cards and a dozen or so punched master name cards in the read hopper, 9-edge first, face down. You are all set to load the program, and process the test cards behind it. To do this, press the load key on the card read-punch. If you have done everything right, your program will be loaded, and the test cards reproduced in a matter of seconds.

To make sure that the cards are correctly reproduced, manually check the cards in the normal stacker on the punch side against the cards in the normal stacker on the read side. When you are convinced that the cards are correct, load the rest of your master card file in the hopper, and press the start key on the card read-punch. A file of 3,000 master cards should take about 12 or 13 minutes to punch.

Program Check-Out

Let's assume the reproduced cards did not match the test cards. Or, the machine went into a loop, with lights flashing all over the console, and nothing was accomplished. Or, perhaps the machine stopped after loading the program, and did not reproduce the first card. In other words, there is a bug (or bugs) in the program.

You, of course, won't know what it is (or they are) until you check out the program. This is an operation generally referred to as *debugging the program.* But, before you check out the program, make sure that the key punch operator correctly punched the three leader cards, the five instruction cards, and the clear and branch card.

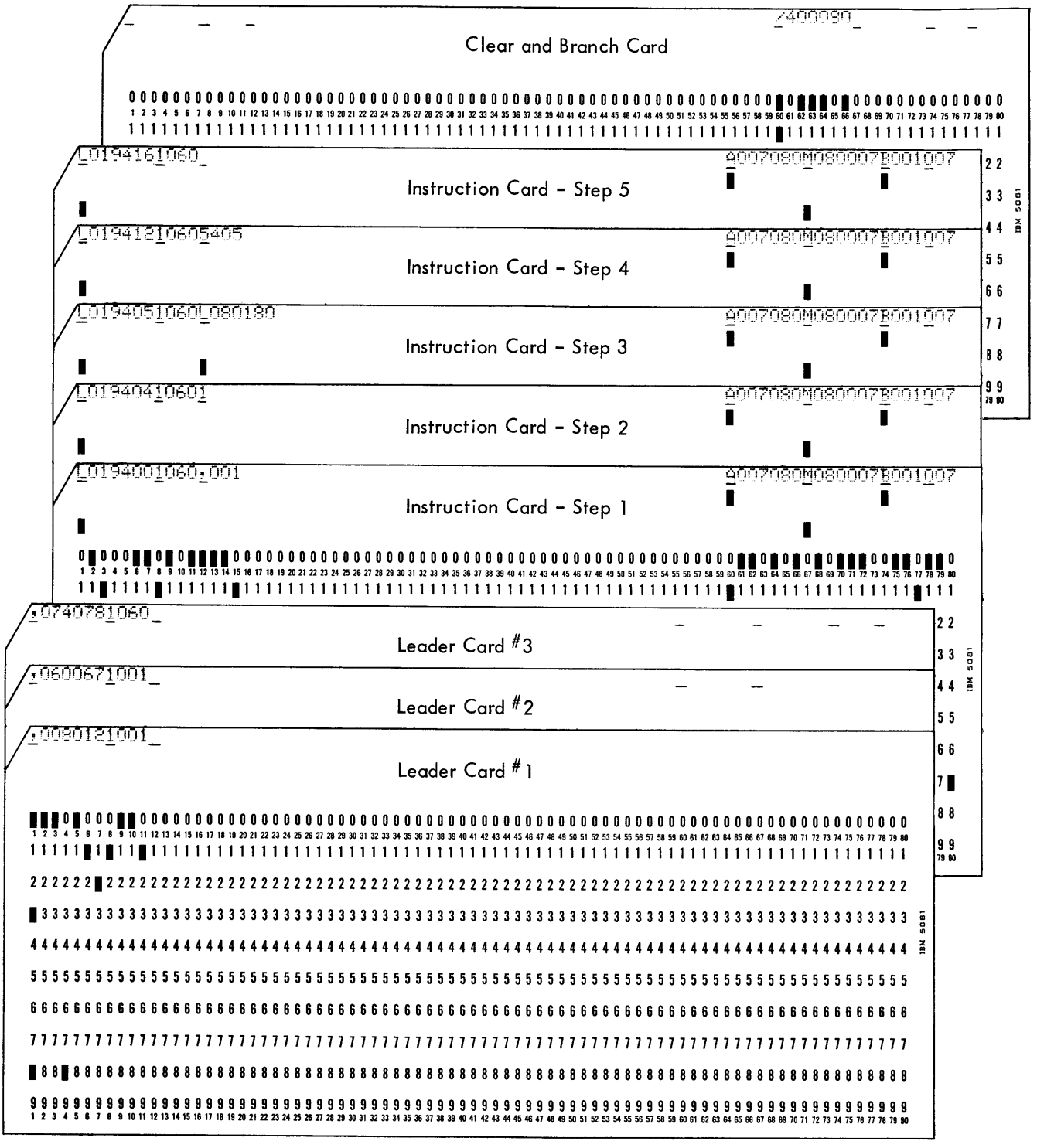
One way to check out the program is to display the characters in locations 400-416, one at a time by use of the console. When you do this, you better know the bit configuration of each character because that's the way the console displays it.

With this method, set the mode switch on CHARACTER DISPLAY and the four manual address dials to 0400. Pressing the start key displays the character in location 0400 in the B-register. The correct character in this location should be a comma with a word mark (,). The bit configuration should be A821M.

The other characters in the program can be displayed one at a time by advancing the manual-address dials to the desired number, and by pressing the start key. An error may be corrected by entering the correct character in the *alter mode* as described in the 1401 reference manual. For obvious reasons, this method is not recommended.

Storage Print-Out

The recommended procedure for debugging a program is to first execute a *storage print-out.* With a 1403 printer (100 print positions) any 100-character block of storage (for example, 001-100, 101-200, 201-300, etc.)



NOTE: Leader Cards 1, 2, and 3 set word marks in positions 8, 12, 60, 67, 74, and 78. The word mark in position 1 is set automatically when the Load Key is depressed.

Instruction Cards 1, 2, 3, 4, and 5 are prepunched in columns 1-4, 8-11, and 60-80 as recommended in the IBM 1401 Reference Manual, Form A24-1403, under PROGRAM LOADING ROUTINE. Columns 5-7 and 12-19 are keypunched from the IBM Program Chart (Figure 21).

The Clear and Branch Card punched /400080 in columns 60-66 clears the read-in area (001-080) and branches to the first instruction in location 400.

Figure 22. Self-Loading Instruction Cards for Problem 1

can be printed out one line at a time by use of this feature. Word marks are indicated by 1's printed on a second line directly beneath the high-order position of each word.

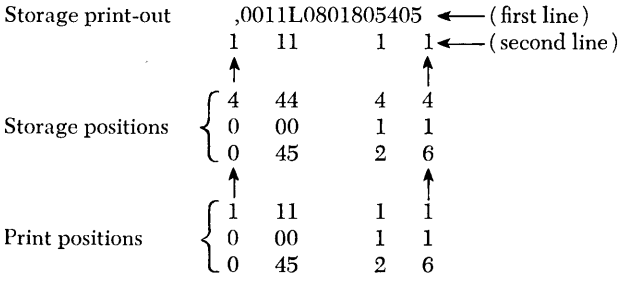
If the 1403 printer has 132 print positions, 132 storage positions will print out on the same line:

Block	*Storage Positions*
00--	0001-0132
01--	0101-0232
02--	0201-0332
03--	0301-0432
04--	0401-0532
etc.	etc.

In *Problem 1,* you are primarily interested in the contents of locations 400-416. To print out the program, proceed as follows:

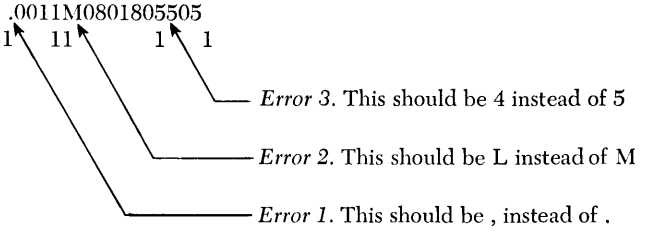
1. Make sure that the printer is in ready status.
2. Set the mode switch to STORAGE PRINT-OUT.
3. Set the hundreds and thousands manual address dials to 03.
4. Press the start key on the console.

The storage print-out for *Problem 1* should look like this:



The 1's on the second line indicate word-mark characters placed in storage by the program loading routine.

Now, let's assume that there are three bugs in the program and the storage print-out looks like this:



Error Analysis

Error 1. The period (operation code for HALT) in location 400 would stop the machine immediately after loading the program. If, after the halt, you pressed the start key, the program would branch to location 001. Because the first character of an instruction word must have a word mark (none there), the process error light and Op register lights would turn on, and stop the machine again. This is often called a *hang-up.*

Error 2. The M instead of L in the position indicated would execute a move operation, starting with the low-order position of the card input area (080), instead of a load operation. The MOVE and LOAD operations transfer data from one location to another. In that respect they are similar. They differ in that the MOVE operation from right to left stops when the first word mark is encountered in either the A- or B-field. The LOAD operation stops when the word mark in the A-field is encountered. Word marks encountered in the B-field during the move are erased.

Stated another way, the instruction L080180 transfers the entire card-read area to the card-punch area. This includes the transfer of the word mark in location 001 to location 101. The M080180 may accomplish the same result *provided* there are no extraneous word marks left in the punch area (101-180) from a previous program. If an extraneous word mark is encountered during the move from right to left, it immediately stops the move. The result is that only the right-hand portion of the read area moves to the punch area. In addition, data to the left of the word mark in the punch area remains undisturbed.

This, then, would be the reason if the reproduced cards did not match the test cards during the test run.

You can prevent the situation in one of two ways:

1. Clear everything in storage before starting the job, or
2. Clear the specific area of storage with an instruction in your program, for example, /180. This instruction would clear the data in locations 100-180, including word marks. Clearing starts at the A-address (180) and continues leftward through the nearest hundreds position. The area is set to blanks, and word marks are cleared.

Error 3. The 5 instead of the 4 in the position indicated would cause a branch to location 505 instead of 405. If there is no word mark in location 505, the machine would stop as for *Error 1.* If the machine finds a word mark, then the character in location 505 (left there from a previous program) would be handled as an operation code.

If the code is illegal, the machine stops and the Op register light turns on.

If the code is legal, that instruction and others immediately following are executed, but this isn't your program. Sooner or later the machine would come to a halt. It might even throw the program into a loop.

Now, this would be a very simple program to check out because all the bugs were caused by key-punch errors — errors that are normally caught and corrected in a verifying operation.

However, suppose there were no key-punch errors, but you had errors in logic. Instead of five instructions,

Step No.	CARD ~~Instr.~~ Address COLUMNS	Instruction OP	A/I	B	Remarks	Effective No. of Characters Inst.	Data	Total
1	1-7	,	008	015	Set Word Marks for steps 2 and 3	7		
2	8-14	,	019	200	Set Word Marks for step 4 and end of step 12	7		
3	15-18	/	199		Clear punch and program area	4		
4	19-25	,	026	033	Set Word Marks for steps 5 and 6	7		
5	26-32	M	080	199	Move program from card area to program area	7		
6	33-39	,	040	182	Set Word Marks for steps 7 and 10 (in program area)	7		
7	40-46	,	047	061	Set Word Marks for step 8 and end of step 9	7		
8	47-53	,	054	196	Set Word Marks for steps 9 and 12 (in program area)	7		
9	54-60	/	182	080	Clear read area and branch to step 10 (in program area)	7		
10	63-69	,	001	189	Set Word Mark in 001 and step 11 (in program area)	7		
11	70-76	M	080	180	Move data from read area to punch area	7		
12	77-80	5	189		Punch-Read and Branch to step 11 (in program area)	4		

Figure 23. Single Card 80-Column Reproduce Routine

suppose you had 500, or 625, or 712? The question will remain unanswered. Suffice it to say, logical errors are often much harder to find than key-punch errors, and your program check-out efforts may take longer than to write the program in the first place. Sometimes, a fellow programmer with more experience can spot logical errors in a program by merely scanning your block diagram.

Clear Storage

It isn't always necessary to clear the whole storage unit before loading a program if you know what to expect and you take care of it in your program. But clearing storage is a good habit to get into. A clear storage routine is described in the 1401 reference manual. The program requires 23 instructions (most of them SET WORD MARKS), but they are all condensed into two cards. In most installations, these cards are prepunched, and are always available for use when required. A storage unit of 16,000 positions can be cleared in just a few seconds.

Single Card 80-Column Reproduce Routine

The master card file in *Problem 1* is reproduced with five instructions as shown in Figure 21. The next time you do this job, try the program shown in Figure 23. This program consists of twelve instructions all punched in a single card. Load this card into the 1401 in front of the cards to be reproduced. Note that the actual reproduce routine is only two steps (11 and 12). All of the other steps merely set word marks for the instructions, clear the necessary areas in core storage, and transfer the routine from the initial read-in to the program area (location 182-199).

Problem 2. Reproducing and Gangpunching

The personnel director has asked your supervisor for an employee master name card file similar to yours, and a printed list of the cards in alphabetical order by surname. The file and list will be used in his office to answer telephone requests for personnel information contained in the cards. In connection with the card file, two things were specifically mentioned by the personnel director which make it different than *Problem 1:*

1. Roll number 41 is to be punched in columns 1-2.
2. The X-punch in column 80 of your master file is to be eliminated, and transferred to column 1 of the reproduced file.

The supervisor discussed the problem with you. You both recognize the possibility of reproducing the master file and preparing the list at the same time. However, you decided against it because it meant sorting your master file alphabetically (it is normally maintained in department and man number sequence) before reproducing, and then re-sorting it back to its original sequence.

So, you both decide to make two jobs out of it, and together you work up a flow chart (Figure 24). This chart shows the sequence of operations of both jobs in graphic form.

The supervisor suggested that you first complete *Problem 2*, then (before starting *Problem 3*,) that you sit down with him to develop some printer specifications for the employees list. The personnel director wanted the name to be shown first on the line, followed by man number, department number, marital status, tax code, and social security number.

Block Diagram

In working up the block diagram for *Problem 2*, (Figure 25), you recognized that this is a combination gangpunch and reproduce operation. Roll number 41 is not in your master cards. Therefore, it must be set up in your program as a *constant*, and moved or loaded into punch positions 101-102 before punching the card. The X-punch (11) in column 80 can be eliminated by simply moving columns 3-79 (storage positions 003-079) to punch positions 103-179. However, the X in 80 must be transferred to punch position 101. You can do this by using a MOVE ZONE instruction (Y080101). The character Y is the operation code for MOVE ZONE. The operand A (080) is where the zone punch (11) is located in the read area. Operand B (101) is where you want it in the punch area. The X will be punched in column 1 of the reproduced card in combination with digit 4 of the roll number. Normally, this combination (11-4) stands for the letter M. In this case, however, it stands for two different characters: digit 4 and zone punch 11.

Your first block diagram is shown in A of Figure 25. Then, remembering some of the difficulty you experienced in *Problem 1* with word marks in the punch area, you think it is a good idea to compare storage locations 003-079 with storage locations 103-179 before punching the card.

In discussing this with your supervisor he points out that the comparing instruction isn't necessary if:

1. You clear the punch area before transferring the information from the read-in area to the punch area, or
2. You transfer the information with a LOAD instruction.

In either case, the information read would always equal the information to be punched. Nothing would be gained by comparing. However, he gives you permission to include the COMPARE instruction in your program provided you follow it with a TEST FOR UNEQUAL instruction, and a HALT AND BRANCH instruction in case of error.

Your second block diagram is shown in B of Figure 25. The COMPARE instruction is C079179. The character C is the operation code for COMPARE. The A-operand 079 is the location of the low-order position of the name card. The B-operand 179 is the location of the low-order position of the data to reproduce.

Following the COMPARE instruction, you test for UNEQUAL. If the comparison is unequal, an error is indicated. In that case, the program branches to a HALT

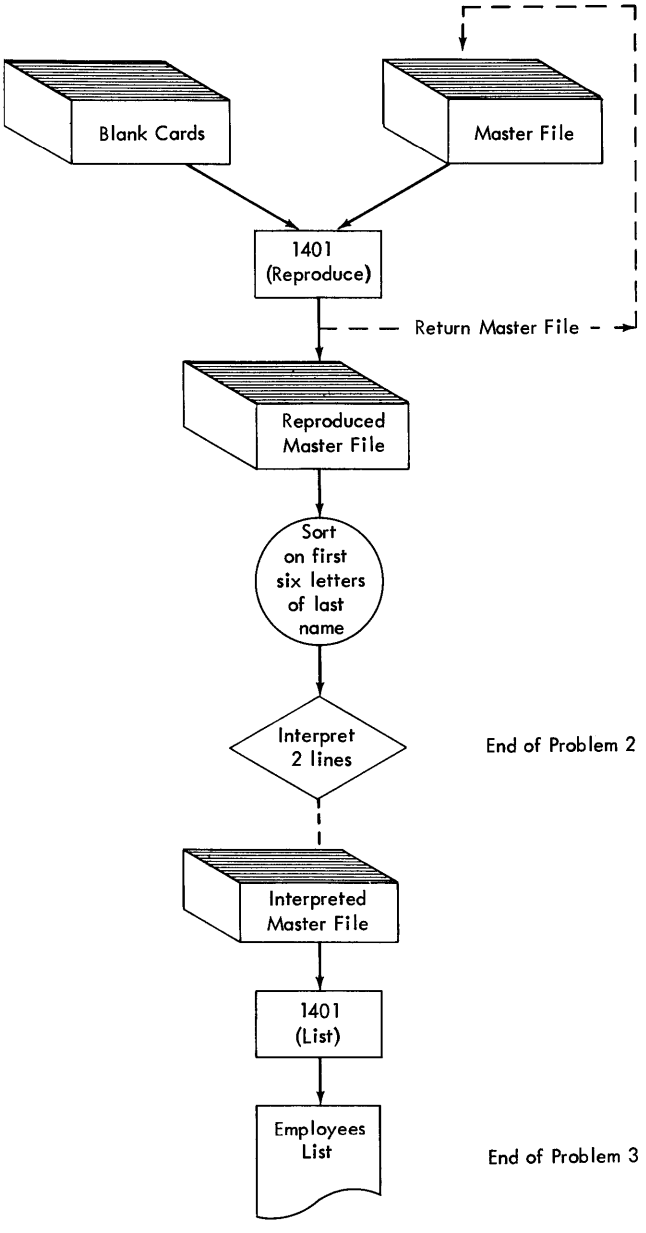


Figure 24. Flow Chart for Problems 2 and 3

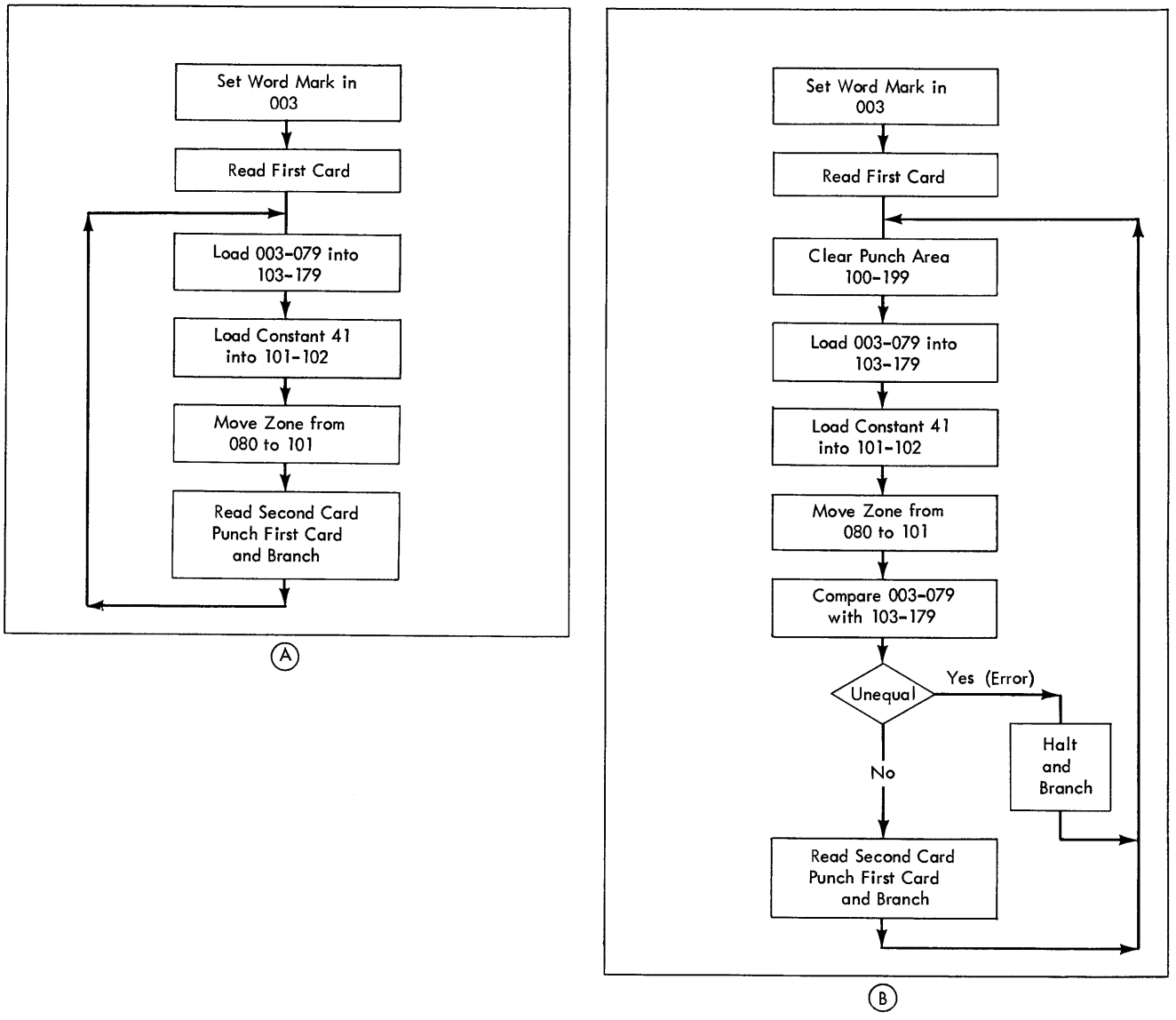


Figure 25. Block Diagrams for Problem 2

AND BRANCH instruction (.xxx). The period (.) is the operation code for HALT, and xxx is the address of the instruction you want to branch to after pressing the start key.

You won't know the address of this instruction until you actually write the program. When an error occurs and the machine stops, you can press the start key to try loading that card a second time. If the error is corrected, the program will continue with the next sequential instruction, and proceed with the next card. If the error persists, the program will stop after each time you press the start key. You are in a loop.

If the comparison is equal, a READ-PUNCH AND BRANCH instruction immediately following the test continues the operation.

The program in B of Figure 25 requires five instructions not used in *Problem 1:*

Instruction

Clear Punch Area – /199
Move Zone – Y080101
Compare – C079179
Branch If Unequal – Bxxx/
Halt and Branch – .xxx

The x's indicate instruction addresses not known at this time. They will be determined as you write the program instructions on the 1401 program chart.

In addition, a constant card is required to gang-punch roll number 41, and a clear-and-branch instruction card at the end of the program. These cards are punched for this example as follows:

	1111	6 666666 6 6677777 777778
Card columns →	1234567890123	0 123456 7 8901234 567890
Constant card →	L002xxx106041	A004080M080004B001011
Clear and branch card		/xxx080

Word marks (as shown) are set by the three leader cards. Operand 002 in columns 2-4 represents the number of characters in the constant word 41 punched in columns 12-13. Operand xxx in columns 5-7 represents at this time the unknown address of the low-order position of the constant 41. You will determine this when you actually write the program.

When the constant card is loaded, the first instruction executed (A004080 in columns 60-66) adds 002 to 011 punched in columns 78-80 of the constant card, to make 013. The next instruction (M080004 in columns 67-73) moves 013 to locations 002-004, and erases 002 in the process. The next instruction (B001 in columns 74-77) branches to the L013xxx instruction starting in location 001. This LOAD instruction will load the constant 41 punched in columns 12-13 (storage locations 012-013) into whatever locations you specify in the B-operand xxx. This you will know when you write the program.

The next instruction (1060) feeds the next card, branches to location 060, and the CLEAR AND BRANCH instruction (/xxx080), punched in columns 60-66, is executed.

For this problem, the sequence of the program cards, therefore, is:

3 leader cards
x number of instruction cards
1 constant card
1 clear and branch card

Figure 26 shows the instructions for *Problem 2.*

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION

IBM 1401 AND 1440 PROGRAM CHART

FORM X24-6437
PRINTED IN U.S.A.

Program: Reproduce Employee Payroll Master Name Card File
(Gangpunch Roll Number 41 in columns 1-2)

Programmer: _____ Date: _____

Step No.	Inst. Address	Instruction OP	A/I	B		Remarks	Effective No. of Characters Inst.	Data	Total
	5-7	12			19	◄----(card columns in instruction cards)			
1	500	,	003			Set Word Mark in 003	4		4
2	504	1				Read first card	1		5
3	505	/	199			Clear punch area	4		9
4	509	L	079	179		Load 079 into 179	7		16
5	516	L	551	102		Load constant 41 in 551 into 102	7		23
6	523	Y	080	101		Move Zone from 080 to 101	7		30
7	530	C	079	179		Compare data in 003-079 with data in 103-179	7		37
8	537	B	546	/		Branch to Halt instruction in 546 if UNEQUAL	5		42
9	542	5	505			Read, Punch and Branch to 505 if EQUAL	4		46
10	546	.	505			Error Stop (Branch to 505 after pressing the start key)	4		50
						CONSTANT			
11	551	4	1			Roll Number 41		2	52
							50	2	52
						NOTE: A-addresses 551 in step 5 and 546 in step 8 were not known until steps 10 and 11 were completed.			
						Clear and Branch Card is punched /500080 in columns 60-66.			

Figure 26. Program Chart for Problem 2

Problem 3. Detail Printing and Adding

The printer specifications, which you and your supervisor agreed upon, for the alphabetical list of employees are graphically shown in the *IBM 1403 Printer Spacing Chart, Form X24-6436* (Figure 27).

First, you decide to use a two-part blank paper-stock form, 8 inches wide, and 8½ inches long. The pin-feed strips on both sides of the form are ½-inch wide. The finished form after the strips are removed, is 7 inches wide and 8½ inches long. The length of the form provides 51 lines of printing (6 lines to the inch).

The specifications in Figure 27 provide for the title:

ALPHABETICAL LIST OF EMPLOYEES---10/13/62
---PAGE xxx

on line 2. This line is indicated by a punch in channel 1, line 2 of the carriage tape. The information to print on this line can be set up in your program as a constant of 48 characters, including the spaces and dashes. The page number xxx will print from a page number counter.

Next, triple-space and set up a 33-character constant (including spaces) to print the first line of the field headings on line 5. Then, single-space and set up a 48-character constant to print the second line of the field headings on line 6. These three lines (title and field headings) are required on all *overflow* sheets as well as on the first sheet.

Next, double-space to line 8. Starting with line 8, twenty master cards will list and double-space. A punch in channel 12, line 46, of the carriage tape indicates that no printing is to occur after line 46, except on the last overflow sheet. When the last master card lists, the carriage skips to line 48, indicated by a punch in channel 2, line 48, of the carriage tape.

On line 48, two predetermined control totals are to print: 3022 750,872,432

These totals will be set up as a 17-character constant in your program. The first total (3022) represents the number of cards in the master card file. The second total (750,872,432) is a *hash* total of the man numbers. These control totals are also machine-developed and printed on line 49 for visual comparison. An EDIT instruction is required to insert the commas where they belong.

The number of pages required for the complete list is 152 (3022 divided by 20).

Refer to the 1401 reference manual for technical details on:

Form design
IBM 1403 printer operating keys, lights and switches
Punching carriage tape
Inserting control tape in carriage
Inserting and positioning the form in the printer

Block Diagram

Figure 28 shows the block diagram for *Problem 3*. As is true with most programs that require many instructions, the sequence of the instructions can vary depending on the wishes of the programmer. Rarely will two different programmers come up with identical programs for the same problem. Regardless of sequence, or the number of instructions used, the finished result must match the specifications set up in the design of the printer form. The more experience a programmer has, and the more shortcuts or tricks he has learned, the shorter his program is apt to be. The less experienced programmer is usually happy with any program he may develop for a given problem as long as it works.

The block diagram shown in Figure 28 is simply one of several approaches to the problem.

In addition to the three leader cards, an x number of instruction cards, and a clear and branch card, you must provide for the following constants and work areas in your program:

ALPHABETICAL LIST OF EMPLOYEES---10/13/62---PAGE
(48 characters)
000 (3-position page number counter)
MAN DEPT. MAR. TAX SOC. SEC.
(33 characters)
N A M E NO. NO. ST. CODE NO.
(48 characters)
– (dash for insertion in Social Security No.)
3022 750,872,432
(17 characters for printing predetermined totals)
0000 (4-position card counter)
000000000 (9-position hash total counter)
, , 0 (edit control word 11 characters)
1 (for increasing page number and card count)

Note: The maximum number of characters in a constant word is 48. The characters are punched in columns 12-59.

Compared to *Problems 1* and *2*, the block diagram in Figure 28 for *Problem 3* is a bit more involved, and of course, requires many more instructions and constants. Notice, however, that out of the 42 blocks in the entire diagram, 18 are MOVE operations with which you are quite familiar.

Three of the blocks involve adding operations to increase the page number and the card count, and to develop the hash total.

Two of the blocks are decision functions indicated by a diamond-shaped symbol. The first is a conditional branch (no cards in the hopper and sense switch A on) to the *End-of-Job Routine*. The second is a conditional branch to the *Heading Routine* when the carriage senses the hole punched in channel 12, line 46, of the carriage tape. If neither of these two conditional branches occur, the program continues with the *Main Routine*.

Before the *End-of-Job Routine* is started, the machine stops, because there are no cards in the hopper. At this

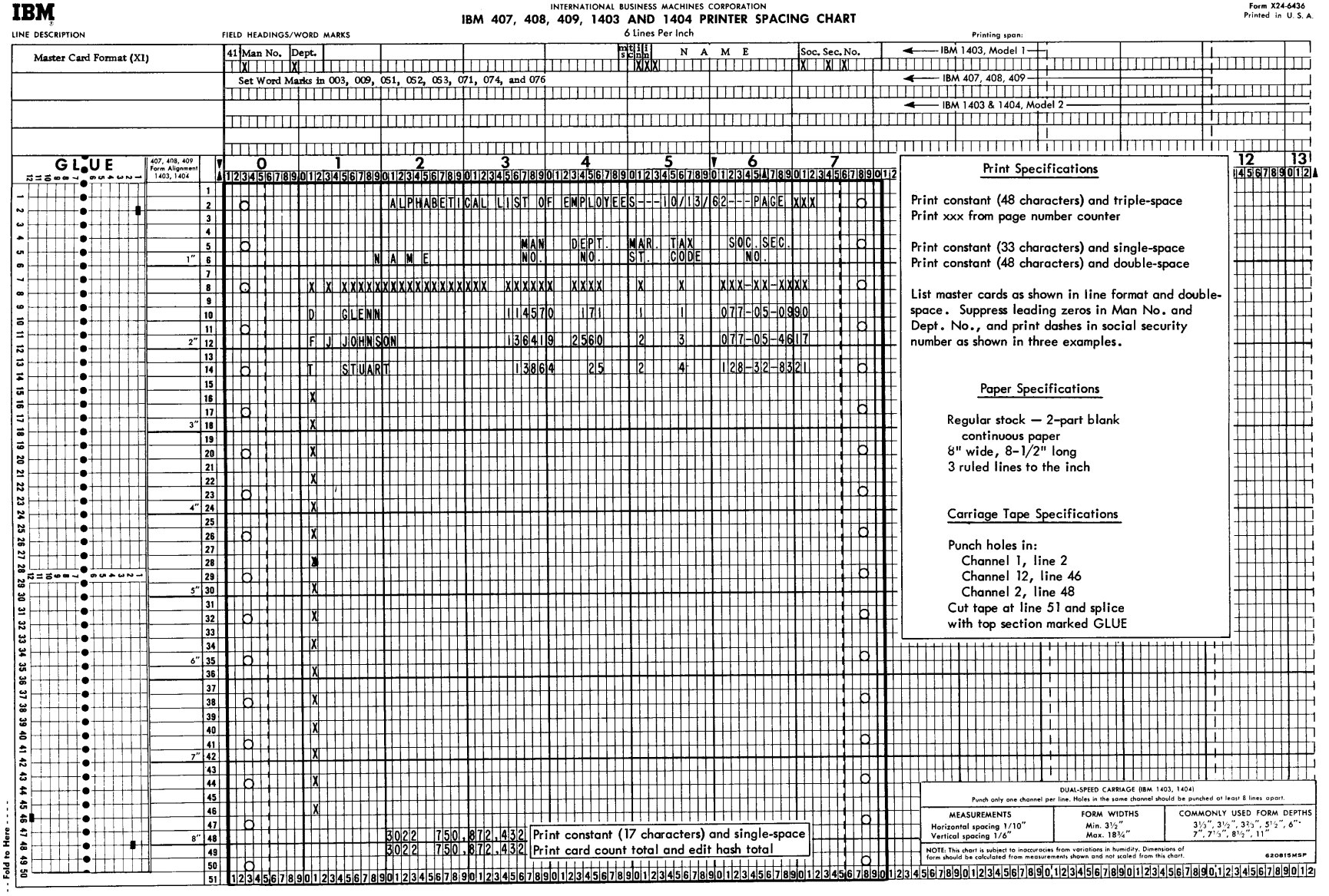


Figure 27. Printer Specifications for Problem 3

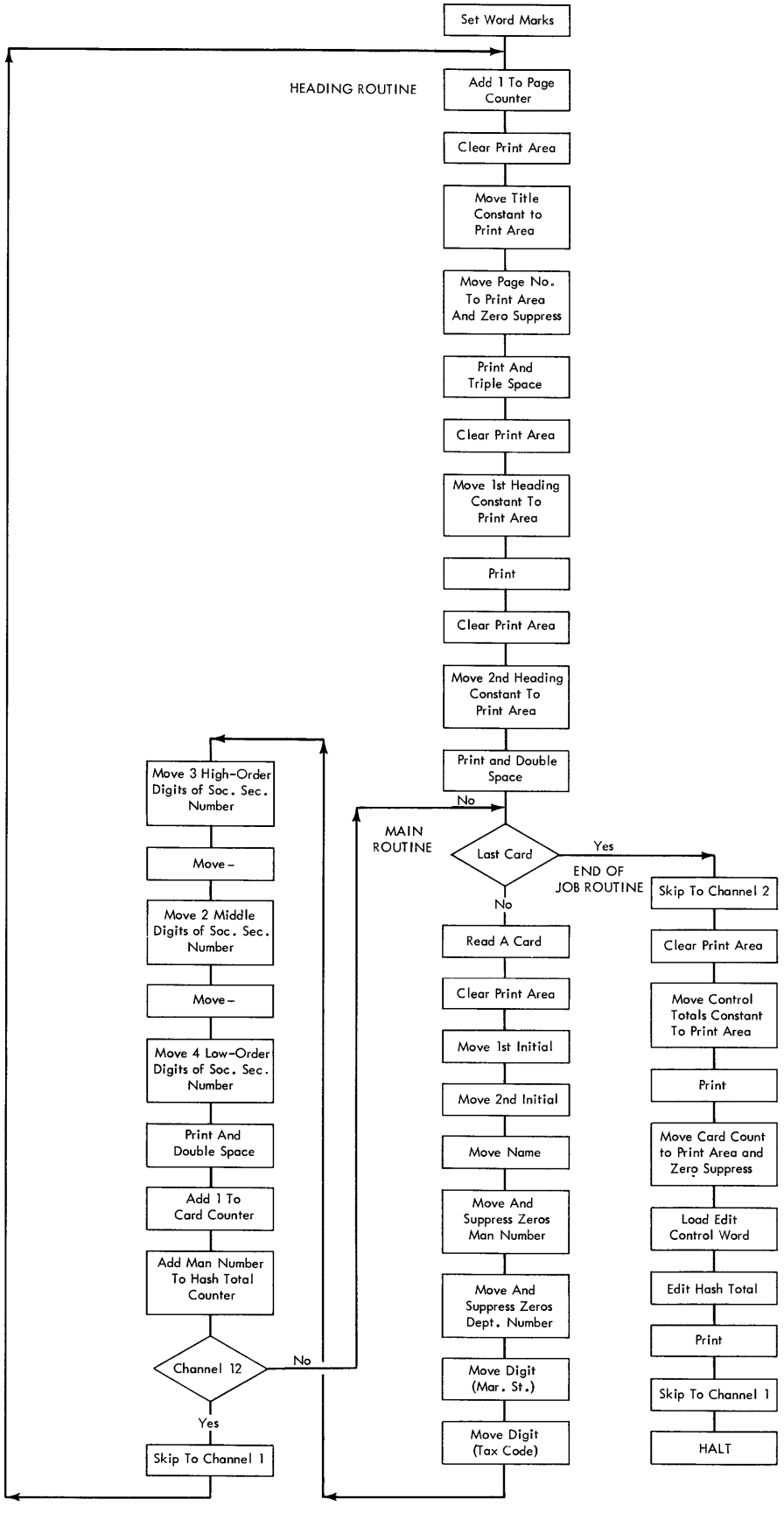


Figure 28. Block Diagram for Problem 3

point (if the job is not completed) you simply put more cards in the hopper, and press the start key. The program will continue with the *Main Routine*. If there are no more cards to put in the hopper because the job is finished, press the start key. The program processes the last card, and automatically branches to the *End-of-Job Routine*. The last instruction in this routine is a HALT (.) instruction to indicate that the report is finished.

Before writing the program on the 1401 program chart, refer to the 1401 reference manual for technical details on the following instructions:

Instruction

Add – Axxxxxx
Last Card Branch – BxxxA
Carriage Channel 12 Branch – Bxxx@
Halt – .
Move and Suppress Zeros – Zxxxxxx
Move Numerical – Dxxxxxx
Move Zone – Yxxxxxx
Load Edit Control Word – Lxxxxxx
Move Characters and Edit – Exxxxxx
Write a Line – 2
Control Carriage Spaces after printing – Fx
Immediate Skip to Carriage Channels – Fx

Figure 29 shows the program for *Problem 3*.

The program in Figure 29 consists of 60 steps as follows:

No. of Steps	*Purpose*	*Storage Area*	*Positions*
4	Initialization Routine	400-427	28
13	Heading Routine	428-481	54
23	Main Routine	482-607	126
10	End-Of-Job Routine	608-646	39
10	Constants	647-821	175
60			422

Notice that the 422 storage positions required by the program as written are consecutive, that is, from 400-821.

Now, if you assume that the program in Figure 29 is correct without checking it, block by block, to the block diagram in Figure 28, you are in for a little surprise. The program has a bug which should be caught and corrected before testing the program. The error is not in the operations specified in the block diagram. Therefore, it is not a logical error. The error is due to the omission of an instruction specified in the block diagram but not provided for in the program chart. See if you can find it.

For purpose of discussion, let's now assume that the omitted instruction should be inserted between steps 20 and 21, and the instruction consists of seven characters. (The actual error is somewhere else). Once you insert this instruction between steps 20 and 21, all addresses from step 21 to step 60 will have to be increased by seven positions. For example, the instruction address for step 21 (now 492) you would have to change to 499, the instruction address for step 22 (now 499) to 506, etc. You would eventually wind up with 61 steps instead of 60.

In addition, all A- and B-operands higher than 492 throughout the whole program, you would have to increase accordingly. For example, step 5, which now reads A821697, would be changed to A828704. Correcting a program by this method, of course, is not recommended because:

1. In the average program you would probably have to change too many instructions.
2. In practice, it is easy to overlook an instruction that should be changed. This causes further difficulty in your program check-out.

The condition just described can be alleviated by leaving gaps between the subroutines so that they are not consecutive. Example: Instead of loading the first instruction in the *Heading Routine* in location 428 as shown in Figure 29, load it in location (say) 450. Instead of loading the first instruction in the *Main Routine* in 482, load it in location (say) 550. Instead of loading the first instruction in the *End-of-Job Routine* in location 608, load it in location (say) 700. And instead of loading the low-order character of the first constant in location 694, load it in location (say) 847.

With this method, in order to bridge the gaps between subroutines, the last instruction in each subroutine must be an unconditional branch to the first instruction of the next subroutine. *Example:* the last instruction in the *Initialization Routine* in Figure 29 (step 4) is in location 421 and reads: ,074076. An unconditional branch instruction should follow in location 428, and should read B450 to branch to the heading routine.

This method requires a few more instructions than a strictly consecutive program. But, in the long run, it pays off because you can insert additional instructions in any subroutine with a minimum number of changes to the rest of the program.

The suggested program, therefore, would include three branch instructions for *linkage*. This increases the number of steps in your program from 60 to 63, and the number of storage positions from 422 to 434.

Example:

No. of Steps	*Purpose*	*Storage Area*	*Positions*
5	Initialization Routine	400-431	32
14	Heading Routine	450-507	58
24	Main Routine	550-679	130
10	End-of-Job Routine	700-738	39
10	Constants	800-974	175
63			434

But, now we run smack into another problem which is often overlooked by programmers. Rule 8 of the *Word Marks* section of the 1401 reference manual reads as follows:

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION

IBM 1401 AND 1440 PROGRAM CHART

FORM X24-6437
PRINTED IN U.S.A

Program: ALPHABETICAL LIST OF EMPLOYEES---10/13/62 Page 1 of 3

Programmer: _____ Date: _____

Step No.	Inst. Address	Instruction OP	A/I	B		Remarks	Effective No. of Characters Inst.	Data	Total
	5—7	12			19	←---(card columns in instruction cards)			
1	400	,	0 0 3	0 0 9			7		7
2	407	,	0 5 1	0 5 2		INITIALIZATION ROUTINE	7		14
3	414	,	0 5 3	0 7 1		Set Word Marks	7		21
4	421	,	0 7 4	0 7 6			7		28
						HEADING ROUTINE			
5	428	A	8 2 1	6 9 7		Add 1 to page counter	7		35
6	435	/	3 3 2			Clear 300-332 print area	4		39
7	439	/				Clear 200-299 print area (Chaining Instruction)	1		40
8	440	M	6 9 4	2 6 8		Move title constant to print area	7		47
9	447	Z	6 9 7	2 7 2		Move page number to print area and suppress zeros	7		54
10	454	F	T			Triple-space after print	2		56
11	456	2				Print title	1		57
12	457	/	2 7 2			Clear title	4		61
13	461	M	7 3 0	2 6 9		Move first field heading constant	7		68
14	468	2				Print first field heading	1		69
15	469	/	2 6 9			Clear print area	4		73
16	473	M	7 7 8	2 6 6		Move second field heading constant	7		80
17	480	F	S			Double-space after print	2		82
						MAIN ROUTINE			
18	482	B	6 0 8	A		Branch to END-OF-JOB ROUTINE if last card	5		87
19	487	1				Read a card if not last card	1		88
20	488	/	2 7 1			Clear print area	4		92
21	492	M	0 5 1	2 1 1		Move first initial of name	7		99
22	499	M	0 5 2	2 1 3		Move second initial of name	7		106
23	506	M	0 7 0	2 3 2		Move last name	7		113
24	513	Z	0 0 8	2 4 0		Move and suppress zeros in man number	7		120
25	520	Z	0 1 2	2 4 6		Move and suppress zeros in department number	7		127

Figure 29. Program Chart for Problem 3 (Part 1 of 3)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION

IBM 1401 AND 1440 PROGRAM CHART

FORM X24-6437
PRINTED IN U.S.A.

Program: ALPHABETICAL LIST OF EMPLOYEES---10/13/62 Page 2 of 3

Programmer: _____ Date: _____

Step No.	Inst. Address	Instruction OP	A/I	B		Remarks	Effective No. of Characters Inst.	Data	Total
	5—7	12			19	◄----(card columns in instruction cards)	127		132
26	527	D	049	251		Move marital status	7		134
27	534	D	050	256		Move tax code	7		141
28	541	M	073	263		Move 3-high order digits of social security number	7		148
29	548	Y	779	264		Move constant (-)	7		155
30	555	M	075	266		Move middle 2 digits of social security number	7		162
31	562	Y	779	267		Move constant (-)	7		169
32	569	M	079	271		Move 4-low order digits of social security number	7		176
33	576	F	S			Double space after print	2		178
34	578	2				Print master card data	1		179
35	579	A	821	800		Add 1 to card counter	7		186
36	586	A	008	809		Add man number to hash total counter	7		193
37	593	B	602	@		Branch to instruction 602 if on channel #12	5		198
38	598	B	482			Branch to MAIN ROUTINE if not on channel #12	4		202
39	602	F	1			Immediate skip to channel #1	2		204
40	604	B	428			Branch to HEADING ROUTINE	4		208
						END-OF-JOB ROUTINE			
41	608	F	2			Immediate skip to channel #2	2		210
42	610	1	271			Clear print area	4		214
43	614	M	796	236		Move control totals constant	7		221
44	621	2				Print control totals	1		222
45	622	Z	800	223		Move card count total and suppress zeros	7		229
46	629	L	820	236		Load edit control word for hash total	7		236
47	636	E	809	236		Edit hash total	7		243
48	643	2				Print card count and hash total	1		244
49	644	F	1			Immediate skip to channel #1	2		246
50	646	•				END-OF-JOB HALT	1		247
							247		

Figure 29. Program Chart for Problem 3 (Part 2 of 3)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION

IBM 1401 AND 1440 PROGRAM CHART

FORM X24-6437
PRINTED IN U.S.A.

Program: ALPHABETICAL LIST OF EMPLOYEES---10/13/62 Page 3 of 3

Programmer: _____ Date: _____

Step No.	Inst. Address	Instruction OP	A/I	B		Remarks	Effective No. of Characters Inst.	Data	Total
	5-7	12			19	20 ◄---(card columns in constants cards)---► 39		2-4	
						CONSTANTS			
						(Lower case (b) in constants means space)			
51	694	ALPHA		ABET		(title line) ICAL LIST OF EMPLOYEES---10/13/62---PAGE	247	48	247 295
52	697	000				(page number counter)		3	298
53	730	MAN	b	bb	DE	(first field heading line) PT.bbMAR.bTAXbbbbSOC.SEC.		33	331
54	778	NbAb		Mb	Eb	(second field heading line) bbbbbbbbbbNO.bbbbNO.bbbST.bbCODEbbbbbNO.		48	379
55	779	-				(dash for insertion in social security number)		1	380
56	796	3022		bb	75	0,872,432 (card count and hash total constant)		17	397
57	800	0000				(card counter)		4	401
58	809	0000		00	00	0 (hash total counter)		9	410
59	820	bbb,		bb	b,	bb0 (edit control word for hash total)		11	421
60	821	1				(digit to increase page numbers and card counts)		1	422
							247	175	422
						NOTE: Clear and Branch Card is punched /400080 in columns 60-66.			

Figure 29. Program Chart for Problem 3 (Part 3 of 3)

8. A word mark must be set in the storage position at the immediate right of the last character of the last instruction in the program.

This problem does not exist in Figure 29 because the last character of the last instruction in the program, (.) in location 646, is immediately followed by the first constant with a word mark in location 647.

In the suggested method, the last character of the last instruction in the program, (.) in location 738, is not followed by a word mark in location 739 because the first constant with a word mark is in location 800. The first letter in this constant is the letter A with a word mark which the machine would treat as an operation code (ADD), and there is no telling what would happen at this point.

So, we have to set a word mark in 739, and we can do it in one of two ways. First, include a SET WORD MARK instruction in your *Initialization Routine* (,739). Or, simply include a card at the end of the *End-of-Job Routine* punched as follows:

```
                          111   6666666 6 6677777777778
Card columns ——→ 123456789012   0123456 7 8901234567890
Last instruc-
  tion card  ——→ L0197391060_   A007080M080007B001007
```

Correcting an Error with a Patch Routine

Now in Figure 29 the program is written consecutively, and we have mentioned that an instruction specified in our block diagram (Figure 28) is not included in our program chart.

Did you find it? The error is between step 17 and step 18. Step 17 in location 480 is an FS (DOUBLE-SPACE AFTER PRINT) instruction. A WRITE instruction (2) in location 482 should follow. This instruction is needed to print the second line of our field heading. How can we insert this instruction without making too many changes in our program?

One method is shown in Figure 30, another in Figure 31. In Figure 30, the correction (in this case, inserting a WRITE instruction) is made by branching from the heading routine at step 16 to a patch routine starting in location 830.

Because step 17 is unfortunately a two-character instruction (FS) in location 480-481, we can't very well fit in a four-character instruction (B830). If we tried it, the B830 instruction would overlap into our main routine, which starts at location 482. For this reason we are forced to go back to step 16, and link in the patch routine as shown in Figure 30.

In Figure 31, the correction is made by changing step 18 from B608A to 2830. This inserts the WRITE instruction and branches to the patch routine. The instructions in the patch routine then are:

```
Step 18a   830   B608A
Step 18b   835   B487
```

Programming Shortcuts

In Figure 29, two of the constants are single characters. The first is a dash (-) in location 779 for insertion in the social security number. The second is a one (1) in location 821 for increasing the page counter and the card counter.

These two constants can be eliminated entirely. In the title constant card (step 51) there are six dashes, and any one of them could be inserted in the social security number by correct addressing. The dashes are in locations 677, 678, 679, 688, 689 and 690. If the dash in location 677 is used, the instructions in steps 29 and 31, (which now read Y779264 and Y779267) would read Y677264 and Y677267.

Similarly, the 1 in the read instruction in location 487 (step 19) can be used to increase the page counter and the card counter. Step 5 (A821697) and step 35 (A821800) would then read A487697 and A487800. This is one of the many shortcuts often used by programmers to reduce the number of steps.

Symbolic Programming

When we write programs in actual machine language as in *Problems 1, 2,* and *3,* data and constant words are referenced by addressing the low-order position of the words, and instruction words by addressing the high-order position. This can easily put bugs in your program if you are not careful, particularly if you are using coded addresses above 999.

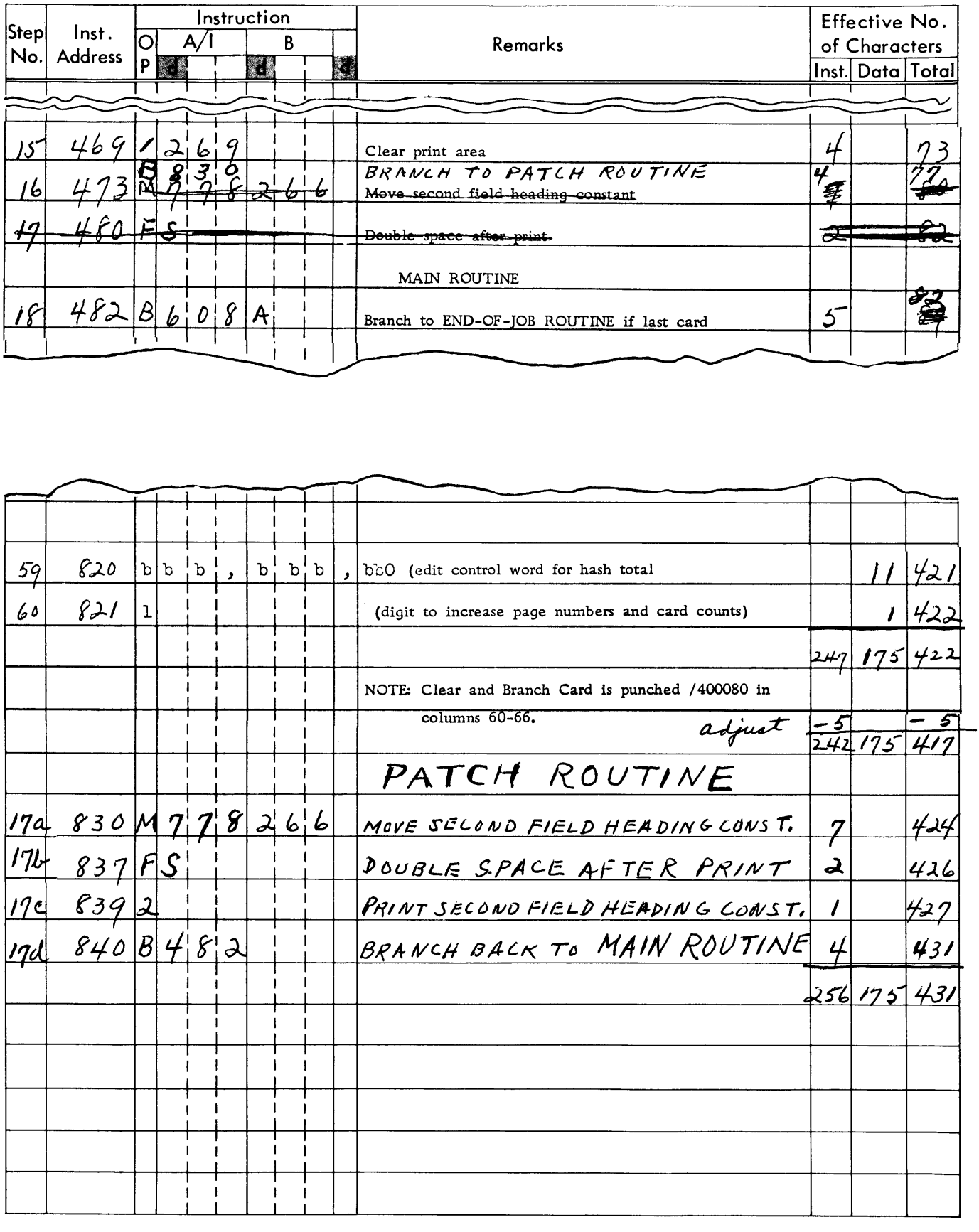
However, very few programs are actually written this way. Once a programmer learns how to program a computer by using actual machine addresses, he quickly resorts to symbolic programming systems. Three of these systems are: *SPS-1, SPS-2,* and *Autocoder.* In these systems, if you have to specify the address of a particular operand or instruction, you use the actual (absolute) address, and not the coded address. *Examples:* Use 1225 instead of S25, 4323 instead of 32T, and 15323 instead of C2C.

But the biggest advantage is that in most cases you don't have to specify addresses at all. In place of actual machine addresses you use *symbols* or *labels.*

Thus, if you move a man number (say: 114570), punched in columns 3-8 of the card, into an area that you label MANNO, you can later move the man number into another area that you label HOLD, and you don't need to worry about the addresses of MANNO and HOLD.

The programmer decides on the labels to use in a particular program. In most cases they are descriptive of the information referenced by them. This is not a requirement, merely a convenience. The only restrictions in connection with labels are that:

1. They do not exceed six letters or digits.
2. The first character must be a letter.



Step No.	Inst. Address	Instruction OP	A/I	B		Remarks	Effective No. of Characters Inst.	Data	Total
15	469	1	2 6 9			Clear print area	4		73
16	473	B M	8 3 0 ~~8 7 8~~	~~2 6 6~~		BRANCH TO PATCH ROUTINE ~~Move second field heading constant~~	4 ~~5~~		77 ~~82~~
~~17~~	~~480~~	~~F~~	~~S~~			~~Double-space after print.~~	~~2~~		~~82~~
						MAIN ROUTINE			
18	482	B	6 0 8	A		Branch to END-OF-JOB ROUTINE if last card	5		~~83~~ 82
59	820	b b b ,	b b b ,			b00 (edit control word for hash total		11	421
60	821	1				(digit to increase page numbers and card counts)		1	422
							247	175	422
						NOTE: Clear and Branch Card is punched /400080 in columns 60-66. adjust	-5		-5
							242	175	417
						PATCH ROUTINE			
17a	830	M	7 7 8	2 6 6		MOVE SECOND FIELD HEADING CONST.	7		424
17b	837	F	S			DOUBLE SPACE AFTER PRINT	2		426
17c	839	2				PRINT SECOND FIELD HEADING CONST.	1		427
17d	840	B	4 8 2			BRANCH BACK TO MAIN ROUTINE	4		431
							256	175	431

Figure 30. Correcting an Error with a Patch Routine (Program Chart in Figure 29)

Step No.	Inst. Address	Op	A/I	B	Remarks	Effective No. of Characters: Inst.	Data	Total
15	469	1	2 6 9		Clear print area	4		73
16	473	M	7 7 8	2 6 6	Move second field heading constant	7		80
17	480	F	S		Double-space after print	2		82
					MAIN ROUTINE			
18	482	2	8 3 0	8 A	PRINT AND BRANCH TO PATCH ROUTINE ~~Branch to END-OF-JOB ROUTINE if last card~~	4 ~~5~~		56 ~~87~~
19	487	1			Read a card if not last card	1		87 ~~88~~
59	820	b	b b , b b b	,	bb0 (edit control word for hash total		11	421
60	821	1			(digit to increase page numbers and card counts)		1	422
						247	175	422
					NOTE: Clear and Branch Card is punched /400080 in columns 60-66.			
					adjust	-1		-1
						246	175	421
					PATCH ROUTINE			
18a	830	B	6 0 8	A	BR TO EOJ ROUTINE IF LAST CARD	5		426
18b	835	B	4 8 7		BR BACK TO MAIN ROUTINE	4		430
						255	175	430

Figure 31. Correcting an Error with a Patch Routine (Program Chart in Figure 29)

In symbolic programming, not only do you use labels of your own choice for most of your operands, but you use mnemonic operation codes that are much easier to remember than actual operation codes. *Mnemonic* means aiding the memory. CS is the mnemonic operation code for CLEAR STORAGE which is much easier to remember than /. SW (SET WORD MARK) is easier to remember than a comma. For a complete list of operation codes, both actual and mnemonic, see Figure 16.

A program written in symbolic language is called a *source program*, and cannot be executed in this form. It is necessary to translate it into machine language by a processor, which is itself a program. The processor translates the source program into an *object program*. The object program may then be used to produce problem results.

Making changes in a program written in actual machine language can be pretty difficult and time-consuming, because of the need for using absolute addresses. In symbolic programs, since we are not concerned with specific storage locations, changes are much simpler and easier to make. After checking out a program, you can easily insert omissions or make corrections to your program by punching the instructions in symbolic, and inserting them in the proper sequence. When you reprocess the source program a second time, the instruction addresses and the addresses of the operands are changed automatically in the second object program.

If you don't want to reprocess the program a second time, you can always correct the errors with a patch routine, using absolute address locations assigned by the processor in the first object program.

We are now ready to learn how to use symbolic programming and apply it to *Problem 3*. First, let us refer to the *IBM 1401 Symbolic Programming Systems, C24-1480*. The two systems described in the manual are: SPS-1 and SPS-2. They are practically identical. The SPS-1 operates with 1401 systems with 1400, 2000, or 4000 positions of core storage. The SPS-2 operates with 4000, 8000, 12000, or 16000 positions, and is designed to provide maximum efficiency for all configurations of equipment.

SYMBOLIC PROGRAMMING CODING SHEET

In either system you use a specially designed coding sheet (instead of the 1401 program chart) when you program in symbolic language. Figure 32 shows the symbolic program for *Problem 3*. A glance at the label field (columns 8-13) shows that 15 labels, ranging in length from three to six characters, are used throughout the program. The labels chosen in Figure 32 are fairly descriptive of the information referenced by them. The complete list is:

Page	*Line No.*	*Label*	*Meaning*
01	040	START	Beginning of the program
	080	HEAD	Beginning of heading routine
02	020	MAIN	Beginning of main routine
03	010	OVERFL	Channel 12 Test
	030	EOJ	Beginning of end-of-job routine
	140		First 32 characters of title (See *Note*)
	150	TITLE	Last 16 characters of title
	160	PGECTR	Page number counter
	170		First 18 characters of 1st heading line (See *Note*)
	180	FIELD1	Last 15 characters of 1st heading line
	190		First 18 characters of 2nd heading line (See *Note*)
	200	FIELD2	Last 30 characters of 2nd heading line
04	010	DASH	Dash (-) for insertion in social security number
	020	CTLTOT	Control totals
	030	CDCTR	Card counter
	040	HASH	Hash total counter
	050	MASK	Edit control word
	060	ONE	One (1) to increase page counter and card counter

Note: Because the maximum number of characters in one constant entry is 32, two entries are required for the title constant (48), two for the first heading constant (33), and two for the second heading constant (48). The first entry is a DCW statement, and the second entry is a DC statement.

For the IBM 1401 Symbolic Programming System, four special types of operations are provided. These control the assembly process, but are never executed in the object program. Three of these are used in Figure 32. They are CTL on page 01, line 010; ORG on page 01, line 030; and END on page 04, line 070.

The CTL (control) entry is used to specify the core-storage size of the 1401. The digit in column 17 specifies the size of the 1401 by which the source program will be assembled. The digit in column 18 specifies the size of the 1401 by which the object program will be executed. To specify the core-storage sizes of the various systems, the digits used in columns 17-18 may range from 1 to 6 as follows:

1=1.4k
2=2k
3=4k
4=8k
5=12k
6=16k

If the CTL entry is omitted, or if columns 17-18 are blank, the SPS-1 processor program takes it for granted that the system has 1400 core-storage positions, and the SPS-2 4000 core-storage positions.

The ORG (origin) entry specifies the beginning address of the object program, in this case, 0400. If

the ORG entry, or if columns 17-20 are blank, the object program will begin at 0333.

The ENDSTART entry at the end of the symbolic program (page 04, line 070) serves the same purpose as the CLEAR AND BRANCH instruction at the end of a program written in actual machine language. Thus, ENDSTART clears the read-in area 0001-0080, and branches to whatever address the processor assigns to the label START (in this case 0400).

In Figure 32, two *comments* entries are made on page 01, line 020, and page 3, line 130. They are normally inserted in a symbolic program to provide descriptive information in the program listing. Cards punched from these entries are identified with an asterisk (*) in column 8. Comments may begin in any position from 9 to 55 of the card. Comments extending beyond column 55 may cause an error during processing. When comments cards are encountered by the processor, they will be reproduced unaltered but will not affect the assembly in any way.

In Figure 32, all the operation codes are entered in columns 14-16 of the coding sheet in mnemonic symbols. However, actual operation codes can be used if the programmer desires. When mnemonic codes are used they must be *left-justified* (start in column 14). When actual codes are used, they must be *right-justified* (must be entered in column 16).

The complete list of the operation codes used in Figure 32 follows:

Instruction	*Mnemonic*	*Actual*
Set Word Mark	SW	,
Add	A	A (See *Note*)
Clear Storage	CS	/
Move Characters to A- or B-Word Mark	MCW	M
Control Carriage	CC	F
Write a Line	W	2
Branch	B	B (See *Note*)
Read a Card	R	1
Move Characters and Suppress Zeros	MCS	Z
Move Zone	MZ	Y
Move Numerical	MN	D
Load Characters to the A-Word Mark	LCA	L
Move Characters and Edit	MCE	E
Halt	H	.

Note: When the mnemonic and actual codes for an instruction are the same, the code may be entered in either column 14 or 16 of the coding sheet.

Operation DCW* in columns 14-17 defines a constant with a word mark in the high-order position. Operation DC * defines a constant without a word mark. Note that constants are entered in columns 24-55. Columns 56-75 are always left blank.

When using the symbolic programming system, you can specify A- and B-addresses in either absolute or symbolic form. Absolute addresses are usually used when you refer to input and output areas, such as the card-read area, the card-punch area, and the print area.

In Figure 32, addresses from 0001 to 0080 refer to the card-read area, and those from 0201 to 0332 to the print area. Note that they are expressed in four digits. This is also true of addresses over 999. Using SPS, we are not required to figure out the three-character form of addresses. As a matter of fact, three-character addresses are indicated as errors in the program listing, and have to be corrected.

In systems with 12,000 or 16,000 positions of core storage, absolute addresses are always expressed in five digits. Unknown addresses of A- and B-operands must be entered in symbolic form, and the symbol must refer to some instruction or constant or work area that has a corresponding label. Otherwise, absence of a corresponding label is indicated as an error in the program listing.

Symbolic Programming Shortcuts

The symbolic program has two constants, (-) and (1), that can be eliminated entirely in a manner similar to the actual machine language program for *Problem 3*. The constants are specified on page 04, line 010 and line 060.

Although we do not know the exact locations of these constants when the symbolic program is written, we do know that there are six dashes (-) in the TITLE constant (page 3, lines 140 and 150). We can determine the location of one of them very easily by starting with the last character of the TITLE constant (E in the word PAGE), and counting four positions to the left. Therefore, the two instructions on page 2, lines 110 and 130, which now read MZ DASH, we would have to change because we no longer have the label DASH. We can refer to the dash in the TITLE by making our instructions read MZ TITLE in columns 14-21 and −004 in columns 23-26.

Similarly, if we omit the constant 1 labelled ONE, we cannot use the instruction A ONE on page 01, line 080, to increase our page count. Nor can we use the instruction A ONE on page 2, line 170, to increase our card count.

One way to address digit 1 in storage is to refer to the R (read a card) instruction that the processor translates to a 1. Although the address of the R instruction is unknown, we do know that it is five positions to the right of the instruction labeled MAIN BEOJA on page 02, line 020.

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program ALPHABETICAL LIST OF EMPLOYEES Page No. 01 of 4

Programmed by _____ Date _____ Identification PROB3

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND	d	COMMENTS
010			CTL 33										
020		* PROG	RAM FOR ALPHABETICAL LIST										OF EMPLOYEES
030			ORG 0400										
040		START	SW 0003					0009					INITIALIZATION
050			SW 0051					0052					ROUTINE
060			SW 0053					0071					SET WORD MARKS
070			SW 0074					0076					
080		HEAD	A ONE					PGECTR					HEADING ROUTINE
090			CS 0332										ADD 1 TO PAGE
100			CS										AND CLEAR STOR
110			MCW TITLE					0268					MOVE TITLE
120			MCS PGECTR					0272					MOVE PGE COUNTER
130			CC									T	3-SP AFTER WRITE
140			W										WRITE TITLE
150			CS 0272										CLEAR PRINT STOR
160			MCW FIELD1					0269					M 1ST LINE FIELD
170			W										W 1ST LINE FIELD
180			CS 0269										CLEAR PRINT STOR
190			MCW FIELD2					0266					M 2ND LINE FIELD
200			CC									S	2-SP AFTER WRITE

Figure 32. Symbolic Program for Problem 3 (Part 1 of 4)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program ALPHABETICAL LIST OF EMPLOYEES Page No. 02 of 4

Programmed by _____ Date _____ Identification PROB3

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND	d	COMMENTS
010			W										W 2ND LINE FIELD
020		MAIN	B EOJ									A	BR TO END-OF-JOB
030			R										READ A CARD
040			CS 0271										CLEAR PRINT STOR
050			MCW 0051					0211					MOVE 1ST INITIAL
060			MCW 0052					0213					MOVE 2ND INITIAL
070			MCW 0070					0232					MOVE LAST NAME
080			MCS 0008					0240					M MANNO-SUP ZERO
090			MCS 0012					0246					M DEPT-SUP ZEROS
100			MCW 0073					0263					M 3-HIGH-SOC.SEC
110			MZ DASH					0264					INSERT DASH
120			MCW 0075					0266					M 2-MID-SOC.SEC.
130			MZ DASH					0267					INSERT DASH
140			MCW 0079					0271					M 4-LOW-SOC.SEC.
150			CC									S	2-SP AFTER WRITE
160			W										WR MASTER CARD
170			A ONE					CDCTR					INCREASE CD CTR
180			A 0008					HASH					ADD MANNO TO HASH
190			B OVERFL									@	BR ON CHANNEL 12
200			B MAIN										NOT 12 BR TO MAIN
091			MN 0049					0251					MOVE MAR.STATUS
092			MN 0050					0256					MOVE TAX CODE

NOTE: The two entries (lines 091 and 092) were inadvertently omitted when the instructions on page 02 were first written. The omission was discovered before assembly. The two instructions were simply added below the heavy line. Before assembly, these two cards will be manually inserted between lines 090 and 100 of page 02.

Figure 32. Symbolic Program for Problem 3 (Part 2 of 4)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program ALPHABETICAL LIST OF EMPLOYEES Page No. 03 of 4

Programmed by _____ Date _____ Identification PROB3

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010		OVERFL	CC									1	SKIP TO CHAN 1
020			B	HEAD									BRANCH TO HEAD
030		EOJ	CC									2	SKIP TO CHAN 2
040			CS	0271									CLEAR PRINT STOR
050			MCW	CTLTOT				0236					M CONTROL TOTALS
060			W										WRITE CTL TOTALS
070			MCS	CDCTR				0223					MOVE CARD CTR
080			LCA	MASK				0236					LOAD EDIT WORD
090			MCE	HASH				0236					EDIT HASH TOTAL
100			W										WR CC & HASH TOT
110			CC									1	SKIP TO CHAN 1
120			H										END-OF-JOB HALT
130		*CONSTANTS											
140	32		DCW	*		ALPH		ABETICAL LIST OF EMPLOYEES-					
150	16	TITLE	DC	*		-10		/132/62---PAGE					
160	03	PGECTR	DCW	*		000							
170	18		DCW	*		MAN		DEPT.		MAR.			
180	15	FIELD1	DC	*		TAX		SOC. SEC.					
190	18		DCW	*		NA		ME					
200	30	FIELD2	DC	*		NO.		NO.		ST.			CODE NO.

Figure 32. Symbolic Program for Problem 3 (Part 3 of 4)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program ALPHABETICAL LIST OF EMPLOYEES Page No. 04 of 4

Programmed by _____ Date _____ Identification PROB3

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010	01	DASH	DCW	*		-							
020	17	CTLTOT	DCW	*		3022		750,872,432					
030	04	CDCTR	DCW	*		0000							
040	09	HASH	DCW	*		0000		000000					
050	11	MASK	DCW	*		, ,		, , 0					
060	01	ONE	DCW	*		1							
070		END	START										

Figure 32. Symbolic Program for Problem 3 (Part 4 of 4)

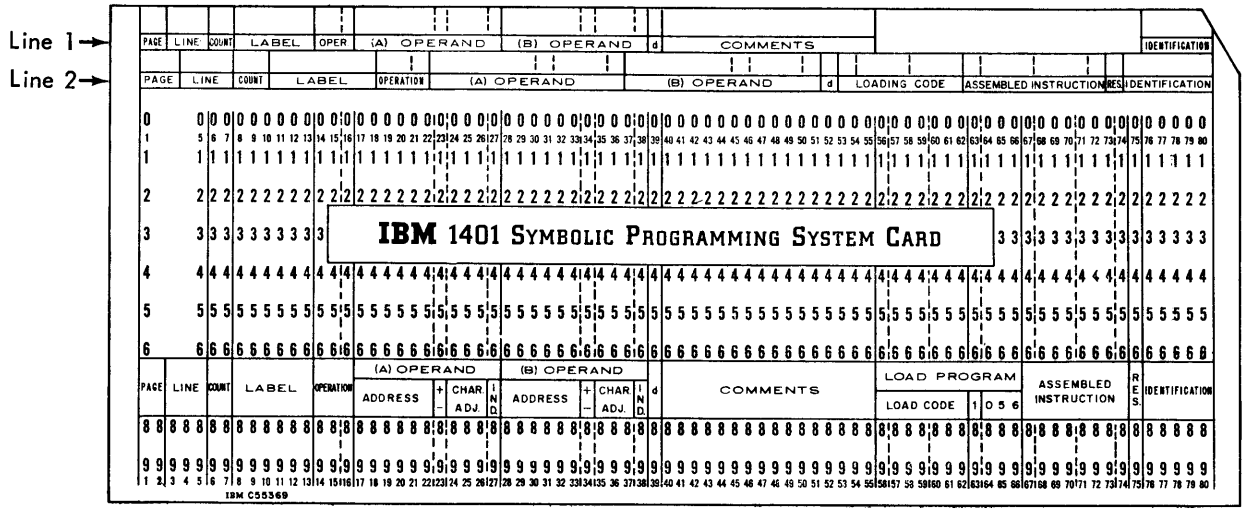


Figure 33. Symbolic-Programming-System Card

In actual machine language this is a conditional branch instruction (B) with a 3-digit A-operand and a 1-character d-modifier. Therefore, the digit 1 is five positions to the right of the instruction code B. The instructions to add 1 to the page counter and the card counter can then be written A MAIN +005.

Using a character-adjusted operand is like referring to a street whose name you do not know. But you do know that it is three blocks past State Street (+003), or two blocks before you come to State Street (−002).

Character-adjusted operands reduce the number of labels required in the source program. However, be careful that insertions do not affect the adjusted operand. Remember, for instance, if an instruction is inserted between the instruction that carries the character you select and the instruction to which you refer by label, then the character adjustment factor must be changed to allow for the insertion.

Example: A 7-character instruction is inserted between lines 020 and 030 on page 02. The instructions to add 1 to the page counter and to the card counter would be A MAIN +012 instead of A MAIN +005.

Punching the IBM 1401 Symbolic Programming System Cards

Entries made on the coding sheets are punched in specially designed cards (Figure 33). Note that the design of this card conforms with the design of the coding sheet. Obviously, page numbers in columns 1 and 2 and program identification in columns 76-80 can be duplicated.

When cards are punched with a printing punch, characters punched in columns 1-55 and 76-80 print on line 1. This lets you sight-verify the interpreted punching on the source program cards with the entries made on the coding sheet before assembly. Columns 56-75 must be left blank on the source program cards. Line 2 is used to interpret the characters on the object program cards with an interpreter after assembly.

If no errors are detected during the verification operation, the source program punched in these cards is ready to be converted to an object program. However, you should make another check from a printed list. The program for preparing the list is one of three standard decks of cards furnished by IBM as follows:

1. Preprocess listing routine
2. SPS processor (deck 1 and deck 2)
3. Postprocess listing routine

Preprocess Listing Routine

The preprocess listing routine is an editing and listing routine that can be used to print the source program before assembly. Its function is to check for possible errors such as invalid addresses, data inadvertently omitted, punching errors, etc. While it is not mandatory to list the source program before assembly, it is wise to make use of this routine to check for coding accuracy and consistency.

OPERATING PROCEDURE

1. Insert a carriage tape punched in channels 1 and 12, and blank continuous form, in the 1403 printer. (See instructions in the 1401 reference manual.) Paper must be at least 11 inches wide.
2. Place the symbolic program deck behind the preprocess listing routine.
3. Place the cards in the card read hopper.
4. Set sense switch A ON, and press the reset button on the 1401 console.
5. Press the load key on the 1402 card-read punch.

The program will automatically restore the printer carriage. All input cards to the program are selected to stacker 1.

ERROR INDICATIONS

When the processor for SPS-2 is used, seven types of errors are indicated in the preprocessing list:

ERROR LEGENDS

ERR 1—Page-line number out of sequence.
ERR 2—Illegal count for DC and DCW cards only. If the count is greater than 32, only columns 24-55 are printed.
ERR 3—Illegal Label.
ERR 4—Illegal mnemonic Op code. In addition, a CTL card in any position other than the first will give this error. These cards are processed as instruction cards.
ERR 5—Illegal A-operand.
ERR 6—Illegal B-operand.
ERR 7—Columns 56-74 not blank.

ERROR ANALYSIS

Figure 34 shows the list prepared with cards punched from Figure 32. The list indicates errors made in coding, or errors made by the key-punch operator. Before the source program can be translated to an object program, all the errors must, of course, be corrected. An explanation of each error as it appears on the list follows:

Page 2, Line 170, ERR 5. This means that the A-operand in this instruction is illegal. Inspection shows that the A-operand ONE is offset one position to the right. This makes the high-order position blank. The first character of any label should always be a letter.

Page 2, Line 091, ERR 1. The card is out of sequence. This card and the following card (line 092) were not inserted between line numbers 090 and 100.

Page 3, Line 030, ERR 3. The label in this instruction is illegal. The first character of any label should always be a letter. In this case, it is blank.

Page 3, Line 050, ERR 6. The B-operand in this instruction is illegal. With absolute addresses, the operands must have four or five digits. The B-operand in this case has three.

Page 3, Line 110, ERR 4. The mnemonic code in this instruction is illegal. In this case, SC should be punched CC.

Page 3, Line 140, ERR 7. Columns 56-74 were not left blank by the key-punch operator.

Page 3, Line 180, ERR 2. The count in this DC card is greater than 32. The count 51 should have been punched 15.

SPS Assembly Procedure

Figure 35 graphically illustrates the procedure for assembling the source program for *Problem 3*. The procedure is as follows:

1. Insert the source program deck between deck 1 and deck 2 of the SPS processor.
2. Load the three decks in the read hopper, and load blank cards in the punch hopper.
3. Press the load key. After the processor cards for deck 1 are loaded into core storage, the program starts immediately to assemble the instructions in the source program deck. During pass 1, deck 1 falls into the normal read stacker, the source deck into stacker 1, and a punched deck (*partial*) into stacker 4.

   At the end of this pass, the processor cards for deck 2 automatically load into core storage, and fall into the normal read stacker behind deck 1. At this time, three cards are punched. The first card is a duplicate of the END card in the source program, and may be discarded. The other two cards contain instructions to clear storage, and fall into the normal punch stacker.
4. When the machine stops, remove the cards from stacker 4 and place them in the read hopper. This *partial* deck is the input to pass 2.
5. Press the start key to resume processing. During this pass, the cards for the object program are punched in a self-loading, one-instruction-per-card format. Each card of the object program will contain the source program statement, the corresponding assembled instruction or data, and the information required to load the assembled instruction. These cards are selected in the normal punch stacker behind the two clear-storage cards mentioned in step 3.

   The third card in this stacker is a *bootstrap* card that initializes the read area with four word marks that are necessary to make the assembled deck self-loading.

Although the object program in the normal punch stacker is theoretically ready to use, the cards are normally listed to disclose any errors that may have occurred in the assembly operation.

One execution of pass 1 and pass 2 is called an *iteration*, and in many cases one iteration is enough to produce an object program. In other cases, two or more iterations are required. This depends on the number of labels used in the source program, and the storage capacity of the 1401 used to assemble the program. The following table shows the number of labels that can be processed for each iteration for the various models.

Processor	*Model*	*Storage*	*Labels Per Iteration*
SPS-1	1	1400	40
	2	2000	100
	3	4000	300
SPS-2	3	4000	260
	4	8000	660
	5	12000	1060
	6	16000	1460

```
                                                                                                                    PAGE  1
PG LIN  CT  LABEL   OP    A OPERAND       B OPERAND       D       LOC  ERROR NOTES  COMMENTS    PROB3

                    CTL   33
 1 020                 * PROGRAM FOR ALPHABETICAL LIST OF EMPLOYEES
 1 030              ORG   0400
 1 040   7  START   SW    0003            0009                                     INITIALIZATION
 1 050   7          SW    0051            0052                                        ROUTINE
 1 060   7          SW    0053            0071                                     SET WORD MARKS
 1 070   7          SW    0074            0076
 1 080   7  HEAD    A     ONE             PGECTR                                   HEADING ROUTINE
 1 090   4          CS    0332                                                     ADD 1 TO PAGE
 1 100   1          CS                                                             AND CLEAR STOR
 1 110   7          MCW   TITLE           0268                                     MOVE TITLE
 1 120   7          MCS   PGECTR          0272                                     MOVE PGE COUNTER
 1 130   2          CC                                    T                        3-SP AFTER WRITE
 1 140   1          W                                                              WRITE TITLE
 1 150   4          CS    0272                                                     CLEAR PRINT STOR
 1 160   7          MCW   FIELD1          0269                                     M 1ST LINE FIELD
 1 170   1          W                                                              W 1ST LINE FIELD
 1 180   4          CS    0269                                                     CLEAR PRINT STOR
 1 190   7          MCW   FIELD2          0266                                     M 2ND LINE FIELD
 1 200   2          CC                                    S                        2-SP AFTER WRITE
 2 010   1          W                                                              W 2ND LINE FIELD
 2 020   5  MAIN    B     EOJ                             A                        BR TO END-OF-JOB
 2 030   1          R                                                              READ A CARD
 2 040   4          CS    0271                                                     CLEAR PRINT STOR
 2 050   7          MCW   0051            0211                                     MOVE 1ST INITIAL
 2 060   7          MCW   0052            0213                                     MOVE 2ND INITIAL
 2 070   7          MCW   0070            0232                                     MOVE LAST NAME
 2 080   7          MCS   0008            0240                                     M MANNO-SUP ZERO
 2 090   7          MCS   0012            0246                                     M DEPT-SUP ZEROS
 2 100   7          MCW   0073            0263                                     M 3-HIGH-SOC.SEC
 2 110   7          MZ    DASH            0264                                     INSERT DASH
 2 120   7          MCW   0075            0266                                     M 2-MID-SOC.SEC.
 2 130   7          MZ    DASH            0267                                     INSERT DASH
 2 140   7          MCW   0079            0271                                     M 4-LOW-SOC.SEC.
 2 150   2          CC                                    S                        2-SP AFTER WRITE
 2 160   1          W                                                              WR MASTER CARD
 2 170   7          A      ONE            CDCTR                          ERR   5   INCREASE CD CTR
 2 180   7          A     0008            HASH                                     ADD MANNO TOHASH
 2 190   5          B     OVERFL                          @                        BR ON CHANNEL 12
 2 200   4          B     MAIN                                                     NOT12 BR TO MAIN
 2 091   7          MN    0049            0251                           ERR 1     MOVE MAR.STATUS
 2 092   7          MN    0050            0256                                     MOVE TAX CODE
 3 010   2  OVERFL  CC                                    1                        SKIP TO CHAN 1
 3 020   4          B     HEAD                                                     BRANCH TO HEAD
 3 030   2   EOJ    CC                                    2              ERR   3   SKIP TO CHAN 2
 3 040   4          CS    0271                                                     CLEAR PRINT STOR
 3 050   7          MCW   CTLTOT          236                            ERR   6   M CONTROL TOTALS
 3 060   1          W                                                              WRITE CTL TOTALS
 3 070   7          MCS   CDCTR           0223                                     MOVE CARD CTR
 3 080   7          LCA   MASK            0236                                     LOAD EDIT WORD
 3 090   7          MCE   HASH            0236                                     EDIT HASH TOTAL
 3 100   1          W                                                              WR CC & HASH TOT
 3 110   2          SC                                    1              ERR   4   SKIP TO CHAN 1
 3 120   1          H                                                              END-OF-JOB HALT
 3 130                 * C O N S T A N T S
 3 140  32          DCW   *     ALPHABETICAL LIST OF EMPLOYEES--          ERR   7
 3 150  16  TITLE   DC    *                    -10/13/62---PAGE
 3 160   3  PGECTR  DCW   *                                 000
 3 170  18          DCW   *                  MAN   DEPT.  MAR.
 3 180  51  FIELD1  DC    *     TAX    SOC.SEC.                               ERR   2
 3 190  18          DCW   *                  N A M E
 3 200  30  FIELD2  DC    *       NO.    NO.    ST.  CODE      NO.
 4 010   1  DASH    DCW   *                                    -
 4 020  17  CTLTOT  DCW   *                  3022  750,872,432
 4 030   4  CDCTR   DCW   *                             0000
 4 040   9  HASH    DCW   *                        000000000
 4 050  11  MASK    DCW   *                        ,  ,  0
 4 060   1  ONE     DCW   *                              1
 4 070              END   START                                  0858

  69 CARDS      15 LABELS    HIGHEST STORAGE ADDRESS USED IS  0858      7 ERRORS
```

Figure 34. Preprocess List of Symbolic Program for Problem 3

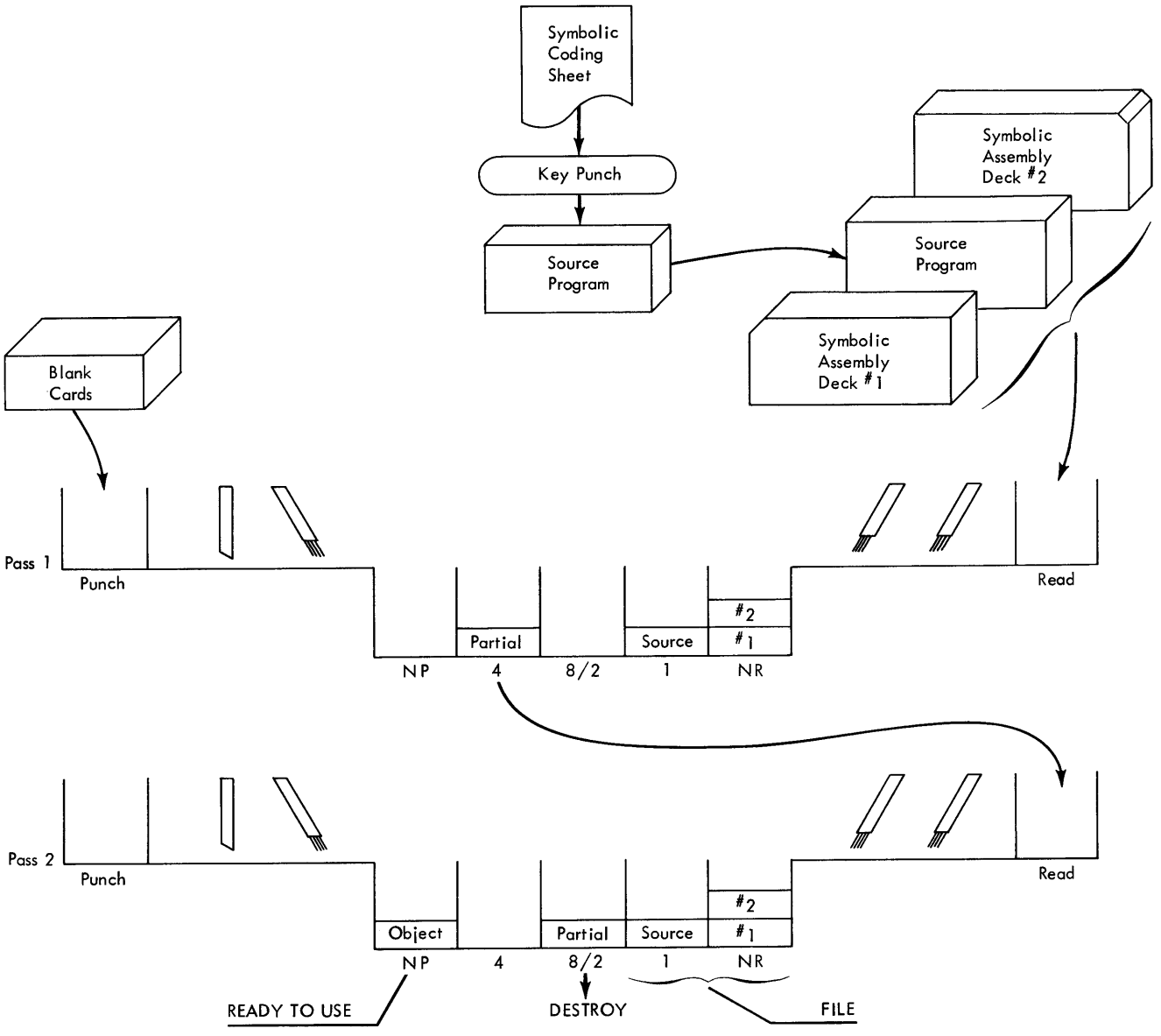


Figure 35. Symbolic-Program-System Assembly Procedure

The model number is punched in the CTL control card as described. The procedure to be followed when additional iterations are required is given in the SPS manual.

Postprocess Listing Routine

The postprocess listing routine can be used to list the object program after assembly is completed. This listing (Figure 36) contains one-instruction-per-card, and each line shows the operation as written originally, and as it appears after translation into 1401 machine language.

OPERATING PROCEDURE

To prepare a postprocess list proceed as follows:

1. Insert carriage tape punched in channels 1 and 12, and blank continuous form, in the 1403 printer. (See instructions in the 1401 reference manual.) Paper must be at least 11 inches wide.
2. Place the object program deck behind the postprocess listing routine deck.
3. Place the cards in the card-read hopper.
4. Set sense switch A ON, and press the check reset and the start reset keys on the 1401 console.
5. Press the load key on the 1402 card-read punch.

The program will automatically restore the printer carriage. All input cards to the program are selected in stacker 1.

```
CLEAR STORAGE 1     ,008015,022026,030034,041,045,053,057073 1026
CLEAR STORAGE 2     L072116,110106,105117B101/I99,027A074028¤027B0010270B026/0991,001/00111710
BOOTSTRAP CARD      ,008015,022029,056063/056029                              ,0240671056


                                                                                                PAGE  1

PG LIN  CT  LABEL   OP    A OPERAND       B OPERAND       D       LOC   INSTRUCTION COMMENTS    PROB3

 1 010          CTL   33
 1 020              * PROGRAM FOR ALPHABETICAL LIST OF EMPLOYEES
 1 030          ORG   0400
 1 040   7  START  SW    0003            0009                    0400  , 003 009   INITIALIZATION
 1 050   7         SW    0051            0052                    0407  , 051 052      ROUTINE
 1 060   7         SW    0053            0071                    0414  , 053 071    SET WORD MARKS
 1 070   7         SW    0074            0076                    0421  , 074 076
 1 080   7  HEAD   A     ONE             PGECTR                  0428  A 822 698   HEADING ROUTINE
 1 090   4         CS    0332                                    0435  / 332       ADD 1 TO PAGE
 1 100   1         CS                                            0439  /           AND CLEAR STOR
 1 110   7         MCW   TITLE           0268                    0440  M 695 268   MOVE TITLE
 1 120   7         MCS   PGECTR          0272                    0447  Z 698 272   MOVE PGE COUNTER
 1 130   2         CC                                      T     0454  F T         3-SP AFTER WRITE
 1 140   1         W                                             0456  2           WRITE TITLE
 1 150   4         CS    0272                                    0457  / 272       CLEAR PRINT STOR
 1 160   7         MCW   FIELD1          0269                    0461  M 731 269   M 1ST LINE FIELD
 1 170   1         W                                             0468  2           W 1ST LINE FIELD
 1 180   4         CS    0269                                    0469  / 269       CLEAR PRINT STOR
 1 190   7         MCW   FIELD2          0266                    0473  M 779 266   M 2ND LINE FIELD
 1 200   2         CC                                      S     0480  F S         2-SP AFTER WRITE
 2 010   1         W                                             0482  2           W 2ND LINE FIELD
 2 020   5  MAIN   B     EOJ                             A     0483  B 609 A     BR TO END-OF-JOB
 2 030   1         R                                             0488  1           READ A CARD
 2 040   4         CS    0271                                    0489  / 271       CLEAR PRINT STOR
 2 050   7         MCW   0051            0211                    0493  M 051 211   MOVE 1ST INITIAL
 2 060   7         MCW   0052            0213                    0500  M 052 213   MOVE 2ND INITIAL
 2 070   7         MCW   0070            0232                    0507  M 070 232   MOVE LAST NAME
 2 080   7         MCS   0008            0240                    0514  Z 008 240   M MANNO-SUP ZERO
 2 090   7         MCS   0012            0246                    0521  Z 012 246   M DEPT-SUP ZEROS
 2 091   7         MN    0049            0251                    0528  D 049 251   MOVE MAR.STATUS
 2 092   7         MN    0050            0256                    0535  D 050 256   MOVE TAX CODE
 2 100   7         MCW   0073            0263                    0541  M 073 263   M 3-HIGH-SOC.SEC
 2 110   7         MZ    DASH            0264                    0549  Y 780 264   INSERT DASH
 2 120   7         MCW   0075            0266                    0556  M 075 266   M 2-MID-SOC.SEC.
 2 130   7         MZ    DASH            0267                    0563  Y 780 267   INSERT DASH
 2 140   7         MCW   0079            0271                    0570  M 079 271   M 4-LOW-SOC.SEC.
 2 150   2         CC                                      S     0577  F S         2-SP AFTER WRITE
 2 160   1         W                                             0579  2           WR MASTER CARD
 2 170   7         A     ONE             CDCTR                   0580  A 822 801   INCREASE CD CTR
 2 180   7         A     0008            HASH                    0587  A 008 810   ADD MANNO TOHASH
 2 190   5         B     OVERFL                          @     0594  B 603 @     BR ON CHANNEL 12
 2 200   4         B     MAIN                                  0599  B 483       NOT12 BR TO MAIN
 3 010   2  OVERFL CC                                      1     0603  F 1         SKIP TO CHAN 1
 3 020   4         B     HEAD                                  0605  B 428       BRANCH TO HEAD
 3 030   2  EOJ    CC                                      2     0609  F 2         SKIP TO CHAN 2
 3 040   4         CS    0271                                    0611  / 271       CLEAR PRINT STOR
 3 050   7         MCW   CTLTOT          0236                    0615  M 797 236   M CONTROL TOTALS
 3 060   1         W                                             0622  2           WRITE CTL TOTALS
 3 070   7         MCS   CDCTR           0223                    0623  Z 801 223   MOVE CARD CTR
 3 080   7         LCA   MASK            0236                    0630  L 821 236   LOAD EDIT WORD
 3 090   7         MCE   HASH            0236                    0637  E 810 236   EDIT HASH TOTAL
 3 100   1         W                                             0644  2           WR CC & HASH TOT
 3 110   2         CC                                      1     0645  F 1         SKIP TO CHAN 1
 3 120   1         H                                             0647  .           END-OF-JOB HALT
 3 130              * C O N S T A N T S
 3 140  32         DCW   *     ALPHABETICAL LIST OF EMPLOYEES--  0679
 3 150  16  TITLE  DC    *                  -10/13/62---PAGE     0695
 3 160   3  PGECTR DCW   *                            000        0698
 3 170  18         DCW   *                MAN   DEPT.  MAR.    0716
 3 180  15  FIELD1 DC    *                  TAX    SOC.SEC.     0731
 3 190  18         DCW   *                N A M E               0749
 3 200  30  FIELD2 DC    *       NO.   NO.   ST.  CODE    NO.   0779
 4 010   1  DASH   DCW   *                                -     0780
 4 020  17  CTLTOT DCW   *                3022  750,872,432     0797
 4 030   4  CDCTR  DCW   *                          0000        0801
 4 040   9  HASH   DCW   *                     000000000        0810
 4 050  11  MASK   DCW   *                     ,  ,  0          0821
 4 060   1  ONE    DCW   *                           1          0822
 4 070          END   START                                     / 400 080
```

Figure 36. Postprocess List of Symbolic Program for Problem 3

POSTPROCESS LIST

Now, compare the locations and the instructions in Figure 36 with those in Figure 29. Because both programs start in location 400, the instruction addresses of the first 17 steps are identical. In Figure 29, the addresses are assigned by the programmer. In Figure 36, they are assigned by the SPS processor.

In Figure 29, a single-character instruction (2) was inadvertently omitted between steps 17 and 18. This required a patch routine (see Figures 30 and 31). The instruction is included in the symbolic program (Figure 32). Therefore, all instruction addresses in Figure 36, (starting with the first instruction in the main routine) are one position higher than those in Figure 29. In all other respects, the two programs are identical.

ERROR INDICATIONS

Errors in the postprocessing list (three types) are indicated as follows:

1. A lozenge (□) printed between page and line number indicates at least one of the following errors on the card.
   a. Undefined symbol in A- or B- operand.
   b. Blank Op code in assembled instructions.
   c. Data to be loaded has been assigned an address in the read area.
   d. A constant has been assigned a count greater than 32.
   e. The count column is blank or zero in the DC, DCW or instruction cards. This will cause a halt in the condensing routine. A blank instruction count cannot occur as a result of assembling. However, it may have been inadvertently left out of patch cards.

2. If a card does not appear to be an assembled card, the unassembled card is printed out with an 80-80 reproduction of the data. This error is most likely to occur if you:
   a. Attempt to list an unassembled deck.
   b. Fail to reiterate a partially assembled deck.
   c. List the output from pass one.

3. The clear storage and bootstrap cards are reproduced 80-80 with a suitable legend at the beginning of the program. If any one of these three cards is missing, or if the cards are not in the proper order, a message appears as follows:

   FIRST CLEAR STORAGE CARD MISSING
   SECOND CLEAR STORAGE CARD MISSING
   NO BOOTSTRAP CARD

It is not necessary, however, that the two clear-storage cards produced during the assembly process be included in the postprocess listing routine.

END-OF-PROGRAM PROCEDURE

When the machine senses an END card that is the last card in the deck, the program halts. You can restart the program by pressing the start key. If the END card is not the last card in the deck, the carriage restores, and control returns to the beginning of the program.

Condensing Program

The condensing program is a separate program supplied with the SPS processor, but is not considered a part of the processor. It is used to convert single-instruction cards in your object program into a condensed deck with several instructions per card.

With only 69 cards in your object program for *Problem 3,* it is hardly worthwhile to condense it into a smaller deck. However, when your program consists of several hundred or several thousand cards, then condensing it is advisable.

Each condensed card will contain up to 38 characters of information of the object program or seven *word-mark locations,* whichever is reached first. In addition, each condensed card will contain enough loading data to make it self-loading. The first card punched by the condensing program will be a self-loading bootstrap card to initialize the read area with appropriate word marks, and to begin the loading process.

As the condensing program reads each card of the assembled object program, the assembled data is transferred into a condensed card-output area. The address of the first word on each card determines the starting location for the card. Conditions that indicate when enough data has accumulated to punch one condensed card are:

1. When the greatest number of characters (not exceeding 38) have accumulated.
2. When seven word-mark locations have been placed in the output area.
3. When a break in the address sequence is reached in the object program due to a DS or ORG card.
4. When an EX (execute) card is encountered.
5. When an END card is encountered.

When any one of these conditions is met, a condensed card is punched. The next instruction or constant from the object program deck becomes the first item of information for the next condensed card.

OPERATING PROCEDURE

1. Place the deck to be condensed (including the bootstrap card) behind the condensing program deck.
2. Place the cards in the card-read hopper.
3. Place blank cards in the punch hopper.

4. Press the reset button on the console (1401).
5. Press the load key on the 1402 card-read punch.

The object deck is selected in stacker 1. The condensed deck stacks in the normal punch stacker.

During the conversion of the object program to a condensed program, the following programmed halts may occur:

1. Each time an END card is processed. Pressing the start key transfers control to the beginning of the routine. After punching an END card, a dummy punch instruction is given to bring the END card into the normal stacker. The dummy card is selected to stacker 8/2.
2. In an instruction or constant card, if any position in columns 57-59 is blank, or if column 57 is not zero. This error, which normally hangs up the routine, cannot occur as a result of assembling but may appear on patch cards.
3. If the count column is blank for instruction or constants. This, too, cannot occur as a result of assembling but may be inadvertently omitted on patch cards.

Pressing the start key after error-halts in items 2 and 3 causes the machine to read a card.

Estimating Time Requirements for Problem 3

In order to properly schedule the work of the IBM 1401 Data Processing System for all jobs, it is often necessary to estimate time requirements for each job.

Problem 3 requires the use of the card reader (1402), the printer (1403), and the processing unit (1401).

OP CODE	OPERATION	CYCLE	RATE	PROCESSING TIME AVAILABLE	
				WITHOUT RELEASE INSTRUCTION	WITH RELEASE INSTRUCTION
1	Read	75 ms 150 ms 225 ms	800 cpm 400 cpm 266 cpm	10 ms 85 ms 160 ms	31 ms 106 ms 181 ms
2	Print — Without print storage Single space Double space Triple space Print — With print storage Single space Double space Triple space	 100 ms 105 ms 110 ms 100 ms 105 ms 110 ms	 600 lpm 572 lpm ** 545 lpm 600 lpm 572 lpm 545 lpm	 16 ms 21 ms 26 ms 98 ms 103 ms 108 ms	
1 and 2	Print & Read With print storage	112.5 ms	533 cpm	8 ms	
3	Print & Read Without print storage	150 ms	400 cpm	18 ms	
4	Punch	240 ms 300 ms 360 ms	250 cpm 200 cpm 166 cpm	22 ms 82 ms 142 ms	59 ms 119 ms 179 ms
5	Read & Punch	240 ms 300 ms	250 cpm/rd 250 cpm/pu 200 cpm/rd 200 cpm/pu	28 ms 22 ms	59 ms 59 ms
6	Print & Punch Without print storage With print storage	 300 ms 240 ms	 200 lpm/pr 200 cpm/rd 250 lpm/pr 250 cpm/rd	 28 ms 20 ms	 57 ms
7	Print, Read, & Punch Without print storage With print storage	 300 ms 240 ms	 200 lpm/pr 200 cpm/rd 200 cpm/pu 250 lpm/pr 250 cpm/rd	 28 ms 20 ms	 57 ms

** Rate used in the estimate

Figure 37. Card and Printer Summary Timing Chart

SYSTEM TIMINGS

Key to abbreviations used in formulas

- L_A = Length of the A-field
- L_B = Length of the B-field
- L_C = Length of Multiplicand field
- L_I = Length of Instruction
- L_M = Length of Multiplier field
- L_Q = Length of Quotient field
- L_R = Length of Divisor field
- L_S = Number of significant digits in Divisor (Excludes high-order 0's and blanks)
- L_W = Length of A- or B-field, whichever is shorter
- L_X = Number of characters to be cleared
- L_Y = Number of characters back to right-most "0" in control field
- L_Z = Number of 0's inserted in a field
- I/O = Timing for Input or Output cycle
- F_m = Forms movement times. Allow 20 ms for first space, plus 5 ms for each additional space
- T_m = Tape movement times
- Σ = Number of fields included in an operation

OPERATION	OP CODE	FORMULA
Add (no recomplement)	A	.0115 (L_I + 3 + L_A + L_B)
Add (recomplement)	A	.0115 (L_I + 3 + L_A + 4 L_B)
Branch	B	.0115 (L_I + 1)
Branch if Bit Equal*	W	.0115 (L_I + 2)
Branch if Character Equal	B	.0115 (L_I + 2)
Branch if Indicator On	B	.0115 (L_I + 1)
Branch if Word Mark and/or Zone	V	.0115 (L_I + 2)
Clear Storage	/	.0115 (L_I + 1 + L_X)
Clear Word Mark	□	.0115 (L_I + 3)
Compare	C	.0115 (L_I + 1 + L_A + L_B)
Control Carriage	F	.0115 (L_I + 1) + F_m
Control Unit	U	.0115 (L_I + 1) + T_m
Divide (aver.)*	%	.0115 (L_I +2 +7 L_R L_Q + 8 L_Q)
Halt	•	.0115 (L_I + 1)
Load Characters to A Word Mark	L	.0115 (L_I + 1 + 2 L_A)
Modify Address*	#	.0115 (L_I + 9)
Move Characters to A or B Word Mark	M	.0115 (L_I + 1 + 2 L_W)
Move Characters and Edit	E	.0115 (L_I + 1 +L_A + L_B + L_Y)
Move Characters to Record or Word Mark*	P	.0115 (L_I + 1 + 2 L_A)
Move Characters and Suppress Zeros	Z	.0115 (L_I + 1 + 3 L_A)
Move and Insert Zeros*	X	.0115 (L_I + 1 + 2 Σ L_A + Σ L_Z)
Move Numeric	D	.0115 (L_I + 3)
Move Zone	Y	.0115 (L_I + 3)
Multiply (aver.)*	@	.0115 (L_I +3 +2 L_C +5 L_C L_M +7 L_M)
No Operation	N	.0115 (L_I + 1)

SYSTEM TIMINGS

OPERATION	OP CODE	FORMULA
Punch a Card	4	.0115 (L_I + 1) + I/O
Read a Card	1	.0115 (L_I + 1) + I/O
Read and Punch	5	.0115 (L_I + 1) + I/O
Select Stacker	K	.0115 (L_I + 1)
Set Word Mark	,	.0115 (L_I + 3)
Start Punch Feed*	9	.0115 (L_I + 1)
Start Read Feed*	8	.0115 (L_I + 1)
Store A-address Register*	Q	.0115 (L_I + 5)
Store B-address Register*	H	.0115 (L_I + 4)
Subtract (no recomplement)	S	.0115 (L_I + 3 + L_A + L_B)
Subtract (recomplement)	S	.0115 (L_I + 3 + L_A + 4 L_B)
Write a Line	2	.0115 (L_I + 1) + I/O
Write and Punch	6	.0115 (L_I + 1) + I/O
Write and Read	3	.0115 (L_I + 1) + I/O
Write, Read and Punch	7	.0115 (L_I + 1) + I/O
Zero and Add	?	.0115 (L_I + 1 + L_A + L_B)
Zero and Subtract	!	.0115 (L_I + 1 + L_A + L_B)

TAPE OPERATIONS

T_m — Tape movement can be determined from the following:

1 = Number of Characters
C = Character Rate

729 II at 200 cpi = .067 ms
at 556 cpi = .024 ms
729 IV at 200 cpi = .044 ms
at 556 cpi = .016 ms
7330 at 200 cpi = .139 ms
at 556 cpi = .050 ms

729 Model II, Read 10.7 + CN ms = TAU interlocked
10.5 + CN ms = Processing interlocked
Write 11.7 + CN ms = TAU interlocked
7.5 + CN ms = Processing interlocked

729 Model IV, Read 6.8 + CN ms = TAU interlocked
6.7 + CN ms = Processing interlocked
Write 7.8 + CN ms = TAU interlocked
5 + CN ms = Processing interlocked

7330 Read 20.5 + CN ms = TAU interlocked
7.7 + CN ms = Processing interlocked
Write 20.3 + CN ms = TAU interlocked
5 + CN ms = Processing interlocked

Rewind
729 Model II = 1.2 minutes/reel
729 Model IV = .9 minutes/reel
7330 (High Speed) = 2.2 minutes/reel

Skip and Blank Tape
(add to subsequent write time)
729 Model II = 40.5 ms
729 Model IV = 27 ms
7330 = 103 ms

Backspace (after Read)
729 Model II = 46 + CN ms
729 Model IV = 33 + CN ms
7330 = 428 + CN ms

Backspace (after Write)
729 Model II = 52 + CN ms
729 Model IV = 37 + CN ms
7330 = 435 + CN ms

Figure 38. Systems Timings

When estimating time requirements, you must consider five factors:

1. Volume of cards to process.
2. Number of lines to print.
3. Speed of the card read-punch.
4. Speed of the printer.
5. Time required to execute each instruction in the program.

Normally, the 1402 reads cards at a rated speed of 800 cards a minute (75 milliseconds per card). The 1403 prints a line at a rated speed of 600 single-spaced lines a minute (100 milliseconds per line).

Obviously, we cannot estimate time at card-read speed, because of the slower speed of the printer. Neither can we estimate time at maximum printer speed, because we don't know yet how long it will take to execute the program, particularly because single-, double- and triple-spacing are required under certain conditions. Other factors to consider include two special features: read release and print storage. These features, when installed in the system, permit greater processing time through overlapping of certain operations. Without these features, 10 milliseconds (ms) are available for processing time during a card-read cycle. During a print cycle, 16 ms are available when single-spacing, 21 ms when double-spacing, and 26 when triple-spacing.

ROUTINE	ADDR.	INSTRUCTION	FORMULA IN MS.	CALCULATION	RESULT	ALLOW PROC. TIME	ACTUAL TIME REQUIRED
Initial.	0400	, 003 009	.0115(L_i +3)	.0115x10	.1150		
	0407	, 051 052		.0115x10	.1150		
(Executed	0414	, 053 071		.0115x10	.1150		
once)	0421	, 074 076		.0115x10	.1150		.4600
Head	0428	A 822 698	.0115(L_i+3+L_a+L_b)	.0115x14	.1610		
	0435	/ 332	.0115(L_i+1+L_x)	.0115x38	.4370		
	0439	/		.0115x102	1.1730		
	0440	M 695 268	.0115(L_i+1+2L_w)	.0115x104	1.1960		
(Executed	0447	Z 698 272	.0115(L_i+1+3L_a)	.0115x17	.1955		
152 times)	0454	F T	.0115(L_i+1)	.0115x3)	.0345		
	0456	2	.0115(L_i+1)+I/O	.0115x2+110	110.0230	26ms	
	0457	/ 272		.0115x78	.8970		110 ms
	0461	M 731 269		.0115x74	.8510		
	0468	2		.0115x2+100	100.0230	16ms	
	0469	/ 269		.0115x75	.8625		
	0473	M 779 266		.0115x104	1.1960		100 ms
	0480	F S		.0115x3)	.0345		
	0482	2		.0115x2+105	105.0230	21ms	105 ms
Main	0483	B 609 A	.0115(L_i+1)	.0115x6	.0690		
	0488	1	.0115(L_i+1)+I/O	.0115x2+75	75.0230	10ms	
(Executed	0489	/ 271		.0115x77	.8855		
3022	0493	M 051 211		.0115x10	.1150		
times)	0500	M 052 213		.0115x10	.1150		
	0507	M 070 232		.0115x44	.5060		
	0514	Z 008 240		.0115x26	.2990		
	0521	Z 012 246		.0115x20	.2300		
	0528	D 049 251	.0115(L_i+3)	.0115x10	.1150		75 ms
	0535	D 050 256		.0115x10	.1150		
	0542	M 073 263		.0115x14	.1610		
	0549	Y 780 267	.0115(L_i+3)	.0115x10	.1150		
	0556	M 075 266		.0115x12	.1380		
	0563	Y 780 267		.0115x10	.1150		
	0570	M 079 271		.0115x16	.1840		
	0577	F S		.0115x3	.0345		
	0579	2		.0115x2+105	105.0230	21ms	
	0580	A 822 801		.0115x15	.1725		
	0587	A 008 810		.0115x25	.2875		105 ms
	0594	B 603 @		.0115x6	.0690		
	0599	B 483	.0115(L_i+1)	.0115x5	.0575		
(Executed	0603	F 1	.0115(L_i+1)+F_m	.0115x3+45	45.0345	45ms	
152 times)	0605	B 428		.0115x5	.0575		45 ms
End-of-Job	0609	F 2		.0115x3+115	115.0345	115ms	
	0611	/ 271		.0115x77	.8855		115 ms
(Executed	0615	M 797 236		.0115x42	.4830		
once)	0622	2		.0115x2+100	100.0230	16ms	
	0623	Z 801 223		.0115x20	.2300		
	0630	L 821 236	.0115(L_i+1+2L_a)	.0115x30	.3450		100 ms
	0637	E 810 236	.0115(L_i+1+L_a+B_b+L_y)	.0115x28	.3220		
	0644	2		.0115x2+100	100.0230	16ms	100 ms
	0645	F 1		.0115x3+30	30.0345	30ms	30 ms
	0647	.	.0115(L_i+1)	.0115x2	.0230		

Figure 39. Time Estimate for Problem 3

To keep this estimate simple, assume that these features are not available in your 1401.

Because our program is comparatively short, we can make a pretty good estimate if we assume that we have plenty of time to execute the program during the time allowed for processing. We can therefore base our estimate on the following calculations:

Number of cards to process		3022
Number of full pages (20 cards per page)		151
Number of lines per full page:		
Title line	1	
Heading lines	2	
Detail lines	20	23
Total number of lines (151 pages)		3473
Number of lines on last page (152)		
Title line	1	
Heading lines	2	
Detail lines	2	
Control total lines	2	7
Total number of lines (152 pages)		3480

Of the 23 lines for a full page, 1 is followed by a triple space, 1 by a single space, and 21 by a double space. This makes an average of 23 double-spaced lines per page. For our problem we can reasonably estimate 572 lines per minute (see Figure 37). The total number of lines (3480) divided by lines per minute (572) equals slightly over 6 minutes for printing alone. To this we must add about 4 minutes for reading the object program and the master card file. We get the 4 minutes by dividing the total number of cards 3092 (1 bootstrap card, 69 program cards, and 3022 master cards) by 800 (speed of card reading). A conservative estimate therefore is about 11 minutes for the complete job. If we were to use a combination WRITE AND READ instruction in our program, we could save approximately 3 minutes. The WRITE AND READ instruction does both operations at the rate of 400 cards a minute.

Processing Time

To make realistic timing estimates for processing, the individual instructions used and the number of data characters involved in each operation must be considered. By using the formulas given in Figure 38, and by applying them to the object program in Figure 36, we can calculate the processing time for *Problem 3* (see Figure 39). The timings are given in milliseconds.

The calculations shown in Figure 39 are summarized:

1. Program loading–75 × 70 cards	= 5250 ms
2. Heading routine–(110+100+105+45) × 152 pages	= 54720 ms
3. Main routine–(75+105) × 3022 cards	=543960 ms
4. End-of-job routine–115+100+100+30	= 345 ms
Total time	604275 ms

$$\frac{604275}{60000} = 10.07 \text{ minutes}$$

The four SET WORD MARK instructions in the initialization routine require only .4600 ms. This time is not included in the summary because these instructions are executed only once for the complete job.

The 45 ms included in the heading routine is required to move the form six spaces from the bottom of one sheet to the first printing line of the next sheet. We get 45 ms as follows:

Space	*Milliseconds*
First	20
Second	5
Third	5
Fourth	5
Fifth	5
Sixth	5
	45

During this time, instructions located in 603, 605, 428, 435, 439, 440, 447, and 454 are executed.

The 115 ms included in the end-of-job routine is required to move the form from the last line printed on page 152 (line 12) to the predetermined control-totals line (line 48). When printing occurs on line 12, double-spacing occurs to line 14. The skip, therefore, from line 14 to line 48 is 34 spaces. We get the 115 ms as follows:

Space	*Milliseconds*
First 8 spaces	55
26 extra spaces at 2.3 ms	59.8
	115

Problem 4. Group Printing

Let's now assume that the key-punch operators have punched all the weekly payroll cards (Figure 6) for the week ending 10/13/62 (roll number 41). Four control totals were established for each department (card columns 9-12), before the cards were punched:

	Card Columns
1. Number of employees (card count)	
2. Hash total of man numbers	3-8
3. Total of hours worked	13-15
4. Total of amount earned	20-24

The first three control totals were established by the department that originates the weekly records. The fourth total was established by the accounting department after multiplying hours by rate for each employee.

Your problem is to prepare a tabulation that accumulates department totals for the four items listed. The tabulation, which will be prepared on blank paper (single-spaced with no field headings), will be used to check the printed totals against the control totals. If the printed totals agree with the control totals, we can assume the key punching is correct. If not, we must locate and correct the errors. Bear in mind that discrepancies may not necessarily be the result of key punch errors. The control totals may be wrong.

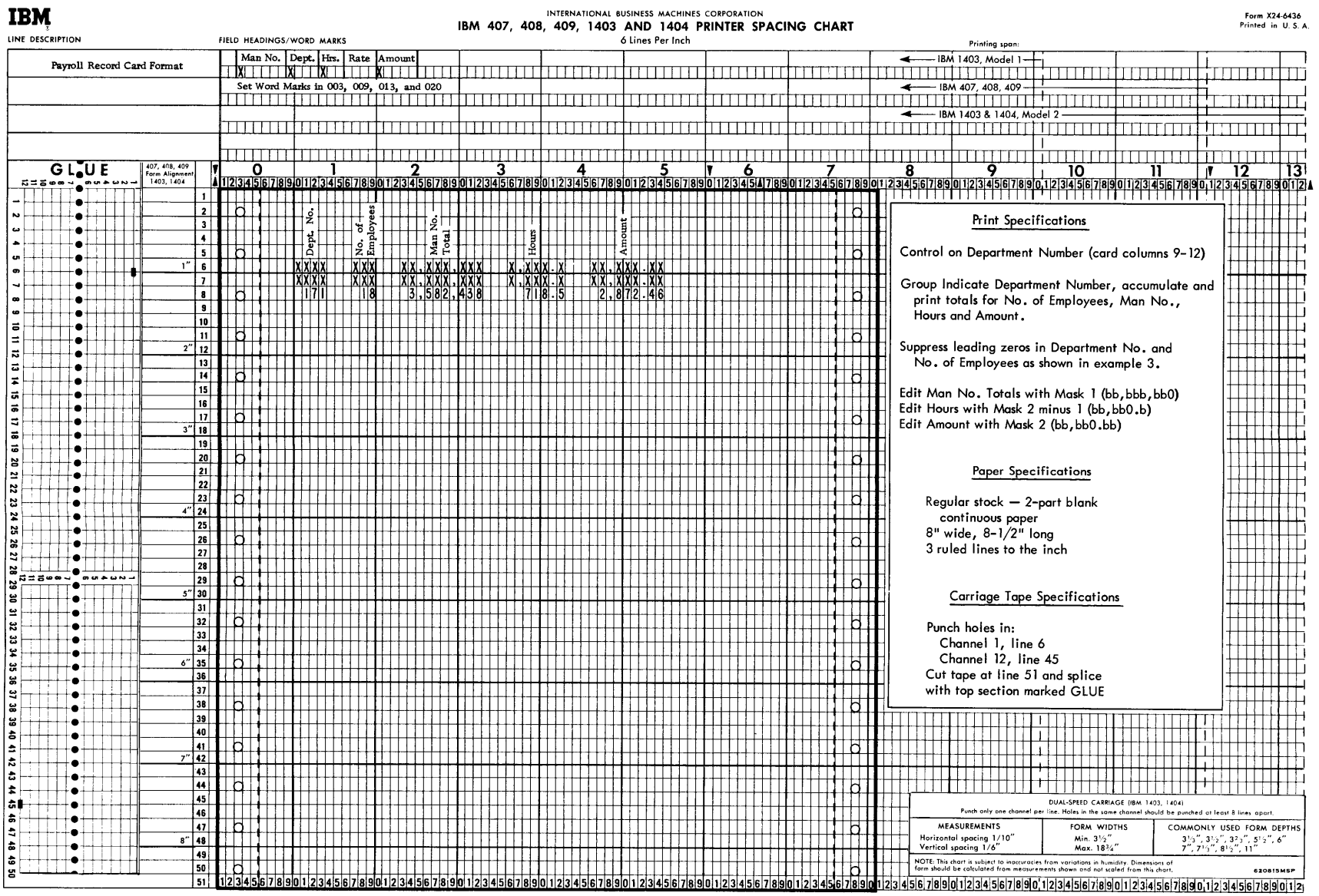


Figure 40. Printer Specifications for Problem 4

Before you can write the program for *Problem 4*, you must do two things:

1. Prepare printer specifications on a 1403 spacing chart (Figure 40).
2. Prepare a block diagram (Figure 41).

Figure 40 specifies the print positions to use for printing the department totals. The specifications provide for 40 lines of single-spaced printing, starting with line 6 of the carriage tape, and ending with line 45. When printing occurs on line 45, the hole in channel 12 of the carriage tape causes overflow to the first printing line of the next sheet.

Figure 41 shows the logic of the program steps to be written in symbolic language. Note that immediately following the READ instruction, a COMPARE instruction is executed. This instruction compares the department number on the card just read with the department number on the preceding card. Initially, this department number is 0000 and is set up with a constant during the loading routine in an area labeled GRPIND (group indicate). Thereafter, the group indicate area will contain the department number of each card after it is processed.

The instructions in the blocks under ROUTE1 are executed card after card until a change in department number is sensed. At this point, the program branches to ROUTE2. The instructions in the blocks under ROUTE2 are executed once for each department to group indicate the department number and print the four totals: Number of employees, man number hash total, hours, and amount.

After the instructions in ROUTE2 are completed, a test on channel 12 tests for overflow. If overflow, a skip to channel 1, and a branch to ROUTE1 follows to process the first card of a new department. If no overflow, the skip is bypassed and the program branches directly to ROUTE1.

When the last card leaves the hopper, the machine stops and the reader light turns on. To process the last card and branch to the end-of-job routine, press the start key. The end-of-job routine first changes the NOP instruction to a HALT, then branches to ROUTE2.Thus, when ROUTE2 is executed for the last group, the program HALT indicates the end of the job.

The symbolic program for *Problem 4* is shown in Figure 42.

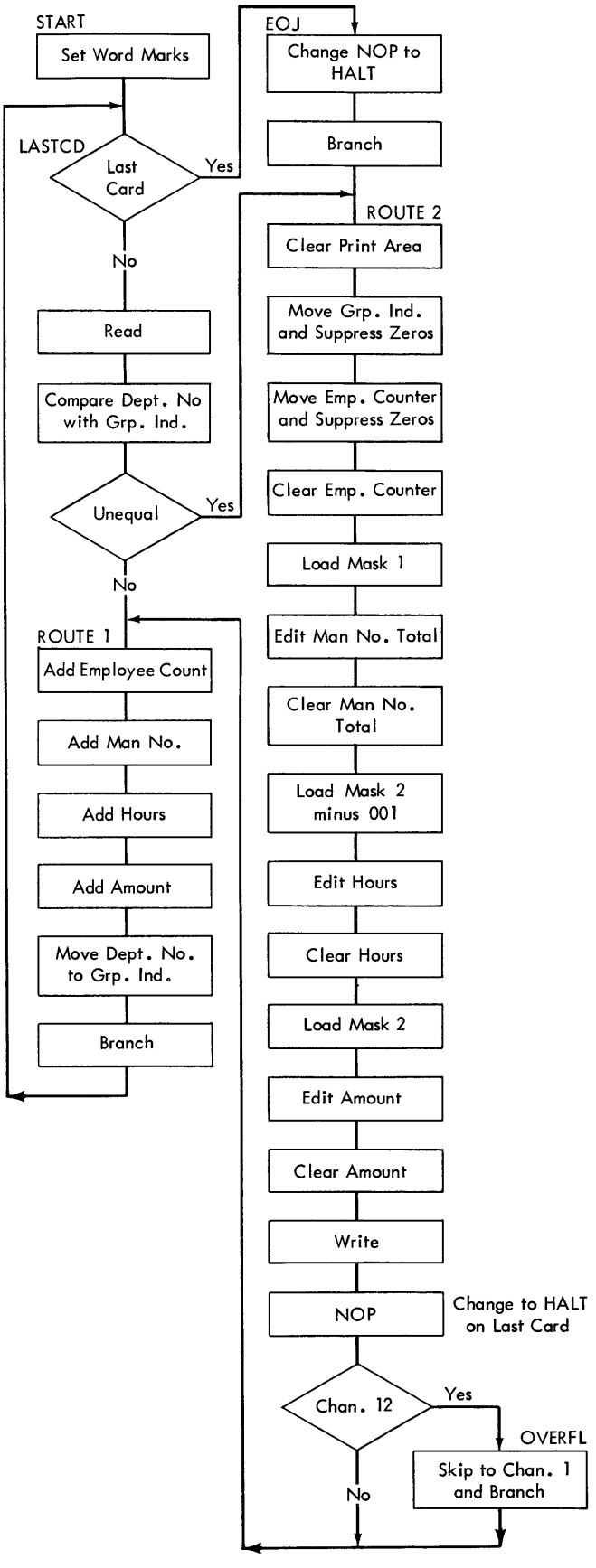


Figure 41. Block Diagram for Problem 4

Problem 5. Multiplication, Selective Listing of Error Cards, Card Punching and Card Selection

Checking to control totals does not necessarily prove the accuracy of key punching because of hidden compensating errors. For example, assume that department 25 had only two employees, with man numbers 114570

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL CHECK RUN Page No. 01 of 02

Programmed by _____ Date _____ Identification PROB4

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010			CTL	33									
020		*CHECK	KR	UNPROBLEM				NO.4 ROLL			N	0	0.41
030			ORG	0400									
040		START	SW	0003				0009					SET WORD MARKS
050			SW	0013				0020					
060		LASTCD	B	EOJ									ASENSE SW A ON
070		READ	R										READ A CARD
080			C	0012				GRPIND					COMP DEPT. NO.
090			B	ROUTE2								/	UNEQ-BR TO RTE2
100		ROUTE1	A	READ				EMPCTR					ADD 1 TO EMPCTR
110			A	0008				MANNO					ADD MAN NO.
120			A	0015				HOURS					ADD HOURS
130			A	0024				AMOUNT					ADD AMOUNT
140			MCW	0012				GRPIND					M DEPT TO GRPIND
150			B	LASTCD									BR ON EQUAL DEPT
160		ROUTE2	CS	0332									CLEAR PRINT AREA
170			CS										IF UNEQUAL
180			MCS	GRPIND				0213					M DEPT TO PRINT
190			MCS	EMPCTR				0219					MOVE EMP COUNTER
200			S	EMPCTR									CLEAR EMPCTR

Figure 42. Symbolic Program for Problem 4 (Part 1 of 2)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL CHECK RUN Page No. 02 of 02

Programmed by _____ Date _____ Identification PROB4

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010			LCA	MASK1				0233					LOAD EDIT CTL WD
020			MCE	MANNO				0233					EDIT MANNO TOTAL
030			S	MANNO									CLR MANNO TOTAL
040			LCA	MASK2	-	001		0242					LOAD EDIT CTL WD
050			MCE	HOURS				0242					EDIT HOURS
060			S	HOURS									CLEAR HOURS TOT
070			LCA	MASK2				0254					LOAD EDIT CTL WD
080			MCE	AMOUNT				0254					EDIT AMOUNT
090			S	AMOUNT									CLEAR AMOUNT
100			W										PRT DEPT TOTALS
110		NOP	NOP										HALT ON LAST CD
120			B	OVERFL								@	BR ON CHANNEL 12
130			B	ROUTE1									NOT 12 BR TO RTE1
140		OVERFL	CC	ROUTE1								1	SK&BR TO ROUTE
150		EOJ	MCW	HALT				NOP					CH NOP TO HALT
160			B	ROUTE2									BR TO ROUTE2
170	04	GRPIND	DCW	*		0000							
180	03	EMPCTR	DCW	*		000							
190	08	MANNO	DCW	*		000	0	0000					
200	05	HOURS	DCW	*		000	0	0					
201	07	AMOUNT	DCW	*		000	0	000					
202	10	MASK1	DCW	*		,		,	0				
203	09	MASK2	DCW	*		,		0.					
204	01	HALT	DCW	*		.							
205			END	START									

Figure 42. Symbolic Program for Problem 4 (Part 2 of 2)

and 118923. They both worked 40 hours. The first employee earned $120.00 (40 hours × 3.000 rate). The second earned $160.00 (40 hours × 4.000 rate). The control totals for department 25 are:

Dept.	*No. of Employees*	*Man No. Total*	*Hours*	*Amount*
25	2	233,493	80.0	280.00

The line of printing in *Problem 4* for department 25 agrees with our control totals. Unfortunately, the keypunch operator punched 160.00 for the first employee and 120.00 for the second. The amount earned for department 25 is still 280.00, yet both cards are wrong.

The purpose of *Problem 5* is to machine-compute the amount earned for each employee, and compare the results with the amount earned that was punched by the operator. If the amounts agree, the payroll card is correct and no action is required. If the amounts do not agree, print a line for each error card as follows:

Dept.	*Man No.*	*Hours*	*Rate*	*Punched Amount*	*Calculated Amount*	
25	114570	40.0	3.000	160.00	120.00	ERROR
25	118923	40.0	4.000	120.00	160.00	ERROR

In addition, we will program the computer to select the error cards in stacker 1, and punch new, corrected cards that will be substituted for the error cards. Card stacking is shown in Figure 43.

Printer specifications, block diagram and symbolic program for *Problem 5* are shown in Figures 44, 45, and 46.

Most of the blocks in Figure 45 are self-explanatory. The main routine multiplies rate by hours, and rounds the result to the nearest cent. The rounded amount is then compared with the punched amount. If the two amounts agree, the error routine is bypassed, and the main routine is repeated for the next card.

If the two amounts are unequal, then the program branches to the error routine, selects the error card in stacker 1, and two output formats (punch and print) are generated. A test is then made for channel 12. If the last line on the form is printed, the hole in channel 12, line 45, of the carriage tape skips the form to the first printing line of the next form. The skip is stopped by the hole punched in channel 1, line 6.

If the form is not on channel 12, the SKIP instruction is bypassed. In either case, a combination WRITE-PUNCH AND BRANCH instruction writes a line for the error card, punches a correct card, and branches back to the main routine.

Multiplication

Two methods of multiplication are available to the programmer. One method, called the *direct method*, involves a multiply-divide special feature described in the 1401 reference manual. When this feature is not installed in the 1401, multiplication can be performed by a subroutine that multiplies by repetitive addition. A multiply subroutine, written in both actual and symbolic language, is contained in the 1401 reference manual. This subroutine can be linked into any program with appropriate branch instructions.

DIRECT METHOD OF MULTIPLICATION

The direct method of multiplication requires that you set up an area somewhere in storage for developing the product. The size of this area varies, depending upon the number of digits in the multiplier and the multiplicand.

Use the following formula to determine the size: *The sum of the number of digits in the multiplier and the multiplicand plus one equals the size of the product area.*

Apply the formula to *Problem 5*. The size of the product area is eight storage positions:

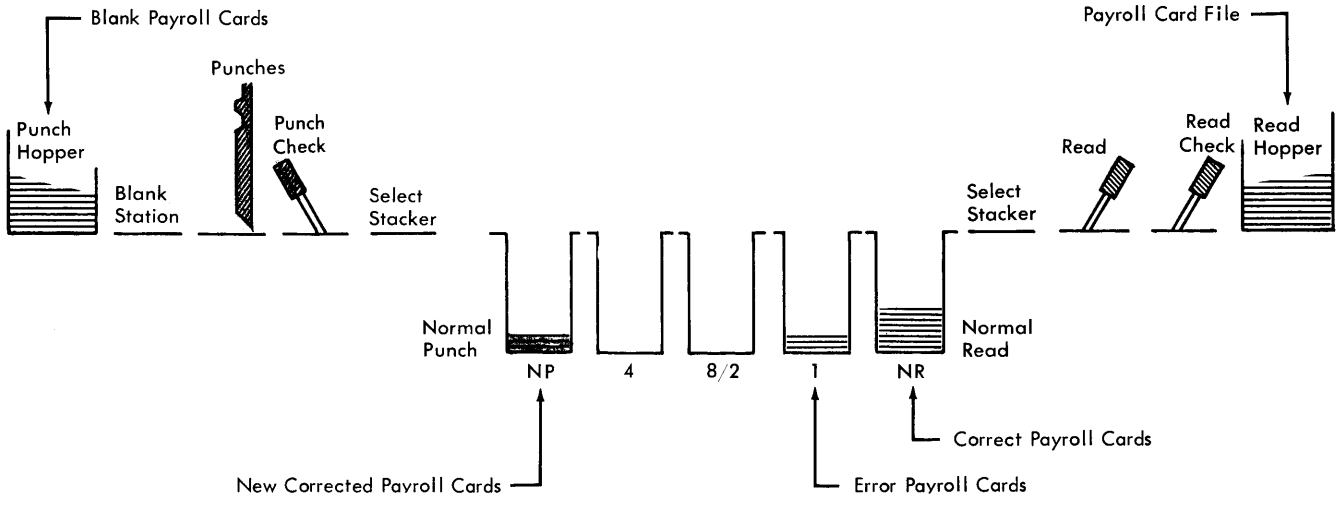


Figure 43. Card Stacking for Problem 5

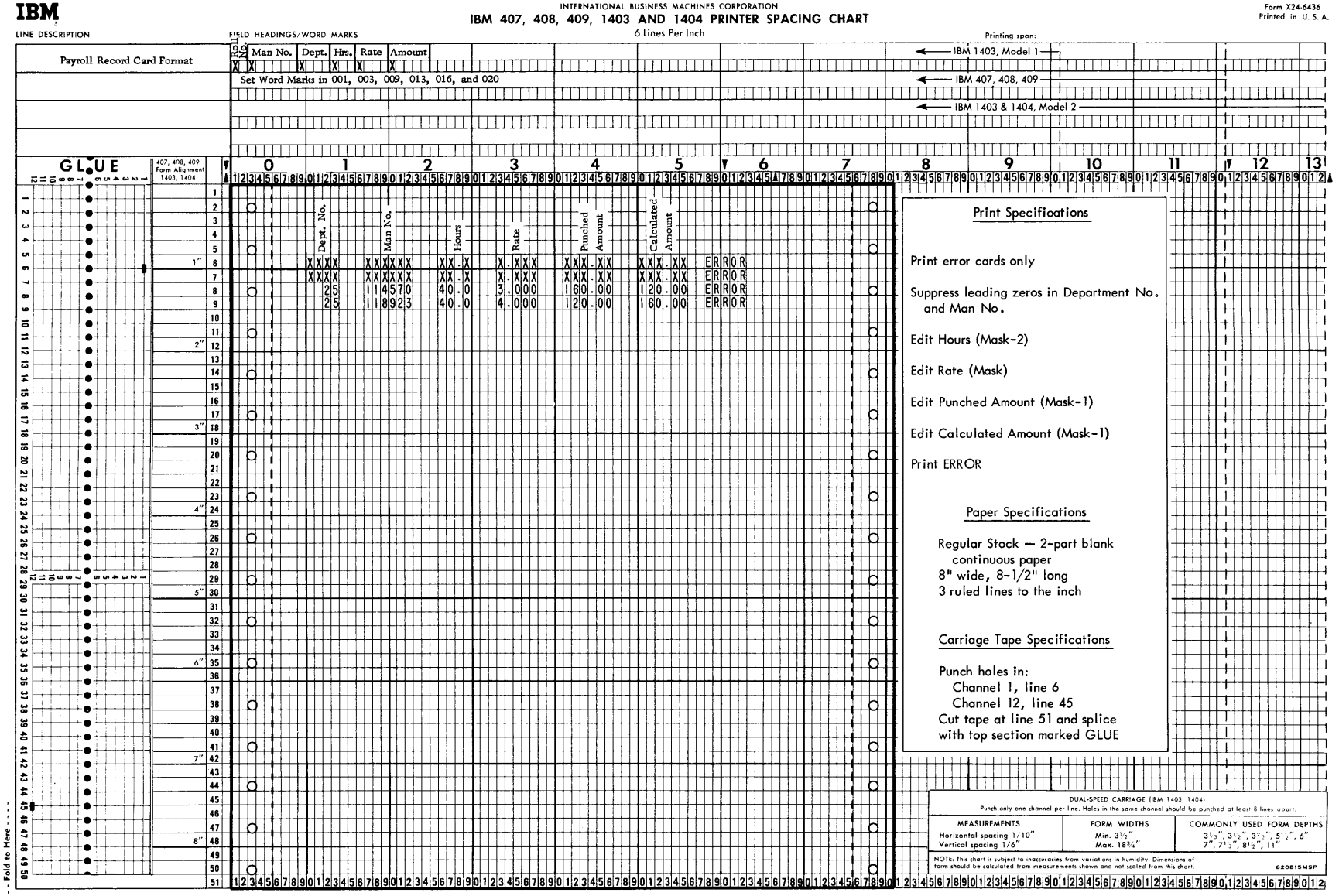


Figure 44. Printer Specifications for Problem 5

Multiplicand (Rate)	*Multiplier (Hours)*	*Product Area (Amount Earned)*
x.xxx	xx.x	xxxx.xxxx

To determine the position of the decimal point in the product *add the number of positions to the right of the decimal points in the multiplicand and the multiplier.* In the example, three positions in *Rate* plus one position in *Hours* equals four positions in the *Amount Earned.*

Once you set up the product area, you can develop a product with two instructions.

First, move the multiplier from locations 0013-0015 to the three high-order positions of the product area with a ZERO AND ADD instruction. In our example, the instruction is:

Operation	*(A) Operand*	*(B) Operand*
ZA	0015	PROD −005

Assume that the A-operand (hours) in locations 0013-0015 is 385. Our product area then reads:

$$\underline{+}38500000$$

Second, execute a MULTIPLY instruction. In our example, the instruction is:

Operation	*(A) Operand*	*(B) Operand*
M	0019	PROD

Assume that the A-operand (rate) in locations 0016-0019 is 2555. The product area at the end of the multiply operation now reads:

$$\underline{0}0983675\ (2.555 \times 38.5 = 98.3675)$$

Note that the multiplier 385 in the three high-order positions of the product area has been destroyed. The *Amount Earned,* therefore, is $\underline{0}098.3675$, which must be rounded to $\underline{0}098.37$. The amount is rounded by adding 5 to the mills position:

$$\begin{array}{rl} & \underline{0}0983675 \\ & \quad\ \ 5 \\ \text{Rounded amount} & \underline{0}0983725 \end{array}$$ Dropped decimals

In our example, the instruction is:

Operation	*(A) Operand*	*(B) Operand*
A	FIVE	PROD −001

The rounded amount is then compared with the punched amount in locations 0020-0024. In our example, the instruction is:

Operation	*(A) Operand*	*(B) Operand*
C	PROD −002	0024

The four instructions required to multiply, round, and compare are entered on page 01, lines 120, 130, 140, and 150, of Figure 46. The program in Figure

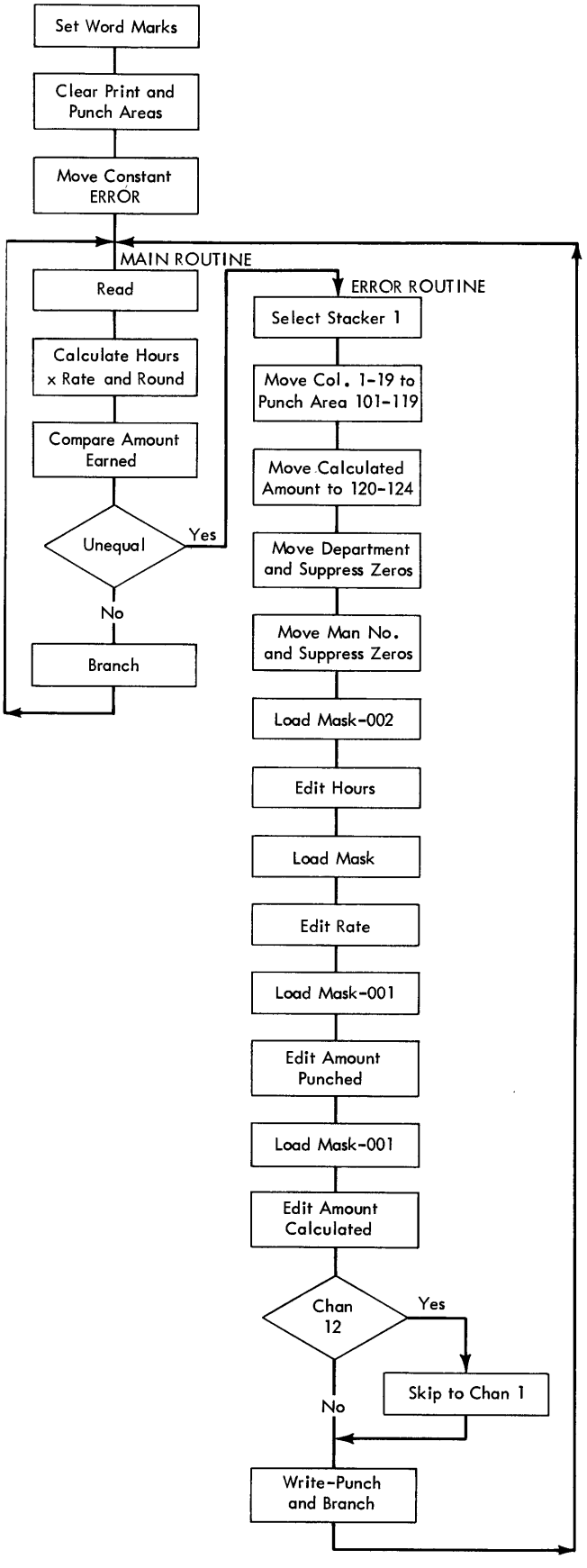


Figure 45. Block Diagram for Problem 5

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL AMOUNT VERIFICATION

Page No. 01 of 02

Programmed by _____ Date _____

Identification PROB5

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010			CTL	33									
020		*PAYROLL	AMOUNT	VERIFICATION									
030			ORG	05000									
040		START	SW	00001				00003					SEW WORD MARKS
050			SW	00009				00013					
060			SW	00016				00020					
070			@S	0332									CLEAR PRINT AND
080			CS										PUNCH AREAS
090			CS										
100			MCW	NOTE				00262					MOVE WORD ERROR
110		MAIN	R										READ A CARD
120			ZA	00015				PROD	-	0005			ZADD MULTPLIER
130			M	00019				PROD					MULT RATE BY HRS
140			A	FIVE				PROD	-	0001			ROUND CALC AMT
150			C	PROD	-	0002		00024					COMPARE AMOUNTS
160			B	ERROR								1	BRTOERROR IFUNEQ
170			B	MAIN									BR TO MAIN IF EQ
180		ERROR	SS									1	SELECT ERROR CD
190			MCW	00019				01199					CHAIN MOVE IN-
200			MCW										STRUCTIONS TO
201			MCW										MOVE COLUMNS

Figure 46. Symbolic Program for Problem 5 (Part 1 of 2)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL AMOUNT VERIFICATION

Page No. 02 of 02

Programmed by _____ Date _____

Identification PROB5

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010			MCW										1-19 TO PUNCH
020			MCW										
030			SW	PROD	-	0006							SET WORD MARK
040			MCW	PROD	-	0002		01234					MOVE CALC AMOUNT
050			MCS	00012				02133					M DEPT SUP ZEROS
060			MCS	00008				02222					M MANNO SUPZEROS
070			LCA	MASK	-	0002		02229					LOAD EDIT WORD
080			MCE	00015				02229					EDIT HOURS
090			LCA	MASK				02237					LOAD EDIT WORD
100			MCE	00019				02237					EDIT RATE
110			LCA	MASK	-	0001		02246					LOAD EDIT WORD
120			MCE	00024				02246					EDIT PUNCHED AMT
130			LCA	MASK	-	0001		02255					LOAD EDIT WORD
140			MCE	PROD	-	0002		02255					EDIT CALC AMOUNT
150			CW	PROD	-	0006							CLEAR WORD MARK
160			B	OVERFL								@	BR ON CHANNEL 12
170		PRINT	WP	MAIN									NOT 12 WP & BR
180		OVERFL	CC	PRINT								1	SK TO CHAN1 & BR
190	05	NOTE	DCW	*		ERROR							
200	08	PROD	DCW	*		00000		00000					
201	01	FIVE	DCW	*		5							
202	07	MASK	DCW	*		0							
203			END	START									

Figure 46. Symbolic Program for Problem 5 (Part 2 of 2)

46 includes five instructions not used in previous problems:

Instructions	*Mnemonic*	*Actual*
1. Zero and Add	ZA	?
2. Multiply	M	@
3. Clear Word Mark	CW	□
4. Write-Punch	WP	6
5. Select Stacker	SS	K

CLEAR WORD MARK

The CLEAR WORD MARK instruction on page 02, line 150, of Figure 46 needs some explanation. First, the adjusted amount in our product area has six digits with a zero and a word mark in the high-order position. Second, the adjusted amount to be printed and punched has five digits. Before moving the adjusted product to the punch and print locations, it is necessary to set a word mark in the product area minus six. After the move, it is necessary to remove the word mark with a CLEAR WORD MARK instruction in preparation for the calculation of the next card.

The four instructions to perform these operations are entered on page 02, Figure 46, as follows:

Line No.	*Oper.*	*(A) Operand*		*(B) Operand*
030	SW	PROD	−006	
040	MCW	PROD	−002	0124
—				
—				
—				
—				
140	MCE	PROD	−002	0255
150	CW	PROD	−006	

Problem 6. Division Subroutine

The XYZ Company buys from other manufacturers many of the component parts that they use in the manufacture of radios and television sets. The company stores the parts in several warehouses, and the Production Department requisitions them as needed.

The cost of each purchased part varies, depending on the quantity purchased, from whom purchased, season of the year, etc. When these items are assembled in the finished products, the amount charged to the production account for each item is based on an average unit cost.

This cost is computed once a week, and a purchased parts inventory report is prepared. This shows the warehouse number, the part number and description, the quantity on hand, the total value of the item based on actual cost, and the average unit cost. The average unit cost is determined by dividing the total actual cost of each item by the quantity on hand.

Another programmer is writing the main program for the purchased parts inventory report. Your supervisor asked you to write a subroutine for computing the average unit cost rounded to the nearest cent. The highest average unit cost will never exceed $999.99.

Your subroutine, which may start at location 0900, must edit the rounded average unit cost into locations 0256-0260, then branch back to an instruction in the main routine labeled PRINT. The divisor (quantity) is punched in columns 51-55, and the dividend (value) is punched in columns 56-62.

Before we attempt to write this routine in symbolic language, let's first do a couple of examples manually:

Example 1

```
      00000.077
24356)01893.46
      1704 92
       188 540 (extra zero)
       170 492
        18 048 (remainder)
```

Rounded Unit Cost:
000.08

Example 2

```
      00009.495
03580)33994.63
      32220
       1774 6
       1432 0
        342 63
        322 20
         20 430 (extra zero)
         17 900
          02 530 (remainder)
```

Rounded Unit Cost:
009.50

Notice that in order to calculate the cost up to the mills position (third position to the right of the decimal point), it is necessary to add an extra zero to the dividend.

Two methods of division are available to the programmer. One method, called the *direct method,* involves a multiply-divide special feature described in the 1401 reference manual. When this feature is not installed in the 1401, division can be performed by use of a subroutine that divides by repetitive subtraction. A divide subroutine written in actual language is contained in the 1401 reference manual. This subroutine can be linked into any program with appropriate instructions.

DIRECT METHOD OF DIVISION

The direct method of division requires that you set up an area somewhere in storage for accepting the dividend and developing the quotient. The size of this area varies, depending upon the number of digits in the divisor and the dividend, plus the number of extra zeros required if the quotient is to be extended. Use the following formula to determine the size: *The sum of the number of digits in the divisor and dividend, plus one, plus one for each extra zero.* Apply this formula to *Problem 6.* The size of this work area is 14 positions:

Divisor (Quantity)	*Dividend (Value)*	*Extra Zeros*	*Work Area*
xxxxx	xxxxx.xx	x	xxxxxxxxxxxxxx

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program DIVIDE SUBROUTINE

Page No. 01 of 1

Programmed by _____ Date _____

Identification PR086

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND	d	COMMENTS
010		* ROUT	INE	FOR CALCUL				ATING AVER				U	UNIT COST
020			ORG	0900									
030			ZA	EXZERO				WKAREA					ADD ZERO & SIGN
040			MCW	0062				WKAREA	-	001			MOVE VALUE
050			D	0055				WKAREA	-	007			COMPUTE UNIT COST
060			A	ROUND				WKAREA	-	006			ROUND UNIT COST
070			LCA	MASK				0260					LOAD EDIT WORD
080			MCE	WKAREA	-	007		0260					EDIT UNIT COST
090			B	PRINT									BR BACK TO MAIN
100	01	EXZERO	DCW	*		0							
110	01	ROUND	DCW	*		5							
120	14	WKAREA	DCW	*		0000	0	00000000		0000			
130	08	MASK	DCW	*				0.					
140													
150													

Figure 47. Divide Subroutine for Problem 6

Line Entry in Figure 47	Instruction OP	(A) OPERAND ADDRESS	(A) OPERAND CH. ADJ	(B) OPERAND ADDRESS	(B) OPERAND CH. ADJ	Function	Work Area
030	ZA	EXZERO		WKAREA		This instruction zero adds a constant zero in the units position of the work area and places a sign for each card calculated. All other positions are reset to zero.	+ 00000000000000
040	MCW	0062		WKAREA	-001	This instruction moves the dividend (0189346) punched in columns 56-62 to the work area minus 1.	DIV. + 0000000 1893460
050	D	0055		WKAREA	-007	This instruction addresses the high-order position of the dividend (work area minus 7) and divides the divisor (24356) into the dividend. An 8-position quotient is developed in the high-order positions of the work area. Notice that the number of positions in the quotient (8) is the same as the number of positions in the actual dividend (7) plus one for the extra zero.	QUOT. + REMAIN-DER + 00000770 18048
060	A	ROUND		WKAREA	-006	This instruction adds a constant "5" to the work area minus six, thus rounding the quotient to the nearest cent.	ADJ. 5 QUOT. + + 00000820 18048
070	LCA	MASK		0260		This instruction loads the edit control word (bbbb0.bb) into the print area (location 0260).	
080	MCE	WKAREA	-007	0260		This instruction edits the adjusted quotient (work area minus seven). Zeros up to the decimal point are suppressed. Thus, for this example, .08 is moved to the print area.	

Figure 48.

The 14-position work area is set up as a constant of 14 zeros with a word mark in the high-order position.

The subroutine for *Problem 6* is shown in Figure 47. Except for the DIVIDE Op code, all other Op codes were used in previous problems. The mnemonic Op code for DIVIDE is D. The actual Op code is % (card code 0-4-8).

FUNCTION OF THE WORK AREA IN A DIVIDE ROUTINE

Figure 48 illustrates the function of the work area. The example uses 24356 as the divisor, and 01893.46 as the dividend. These figures are the same as those used in *Example 1.*

Problem 7. Payroll Register

This problem involves preparing a payroll register, using five different types of cards:

Type of Card	*Card Type Code*
1. Date	D
2. Employee name (same as Figure 18)	-
3. Year-to-date	&
4. Deduction	1
5. Payroll (same as Figure 6)	blank

The formats for the payroll register and the five types of cards are shown in Figure 49. The card type code used to distinguish each type of card is punched in column 80.

The date card (punched as in Figure 49) is to precede the file. The file to be processed was originally four separate files. They were merged (with a collator) in the order listed, in department and man-number sequence. During the merging operation, employee name cards and deduction cards for employees who had no payroll card were selected as unmatched cards, and therefore were not included in the merged file.

New employees, of course, have no year-to-date card, and a deduction card is included only for those employees who have authorized deductions for community chest and/or hospital insurance. Thus, a man-number group may consist of two, three, or four cards. The employee name card is always the first card of the group, and the payroll card is always the last.

In addition to the payroll register, we are to punch a new year-to-date card for each employee included in the payroll register.

Payroll Register Specifications

A preprinted payroll register form (as designed in Figure 49) is used. The form is 12½ inches wide (left to right) and 8½ inches long (top to bottom). The week-ending date, roll number, and department number are to print on line three of each form. This line is specified by a hole in channel 1, line 3, of the carriage tape. The first body line of the payroll register is specified by a hole in channel 2, line 7. Body lines are to be single-spaced. The last body line is specified by a hole in channel 12, line 46. The predetermined line for department totals is specified by a hole in channel 3, line 48.

Program Halts

The program must include instructions to stop the machine under the following conditions:

Initialization Routine

Halt 1 = No date card

Halt 2 = First file card is not a name card

Main Routine

Halt 3 = First card of a man group after a control break is not a name card

Halt 4 = Card has other than -, &, 1, or blank as a card type code in column 80

Final Halt = End of job

Card Stacking

The program must provide for stacking cards as shown in Figure 50.

Block Diagram

The block diagram shown in Figure 51 is one of several approaches to this problem. First note that an initialization routine sets word marks and tests for the presence of the date card and the first name card. A second initialization routine prints the date line on the first form, skips to channel 2, loads the first name into the print area, and calculates the exemption amount. These two routines originate in locations 0200 and 0100, respectively. They are used only once in the operation and are cleared after execution.

The main routine originates in location 0400, and is used repeatedly for all other cards in the file. The routine tests for the four types of cards (employee name,

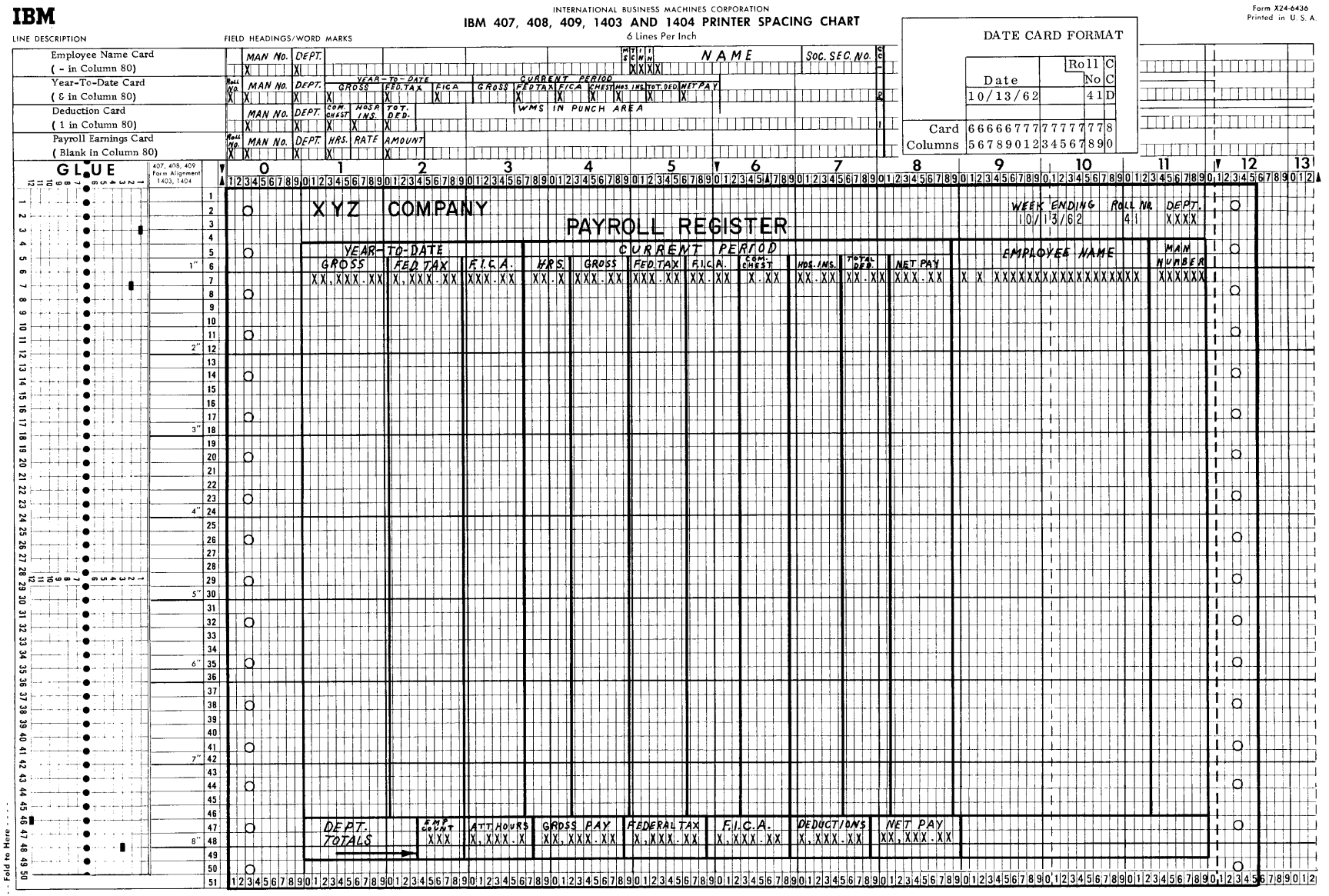


Figure 49. Printer Specifications for Problem 7

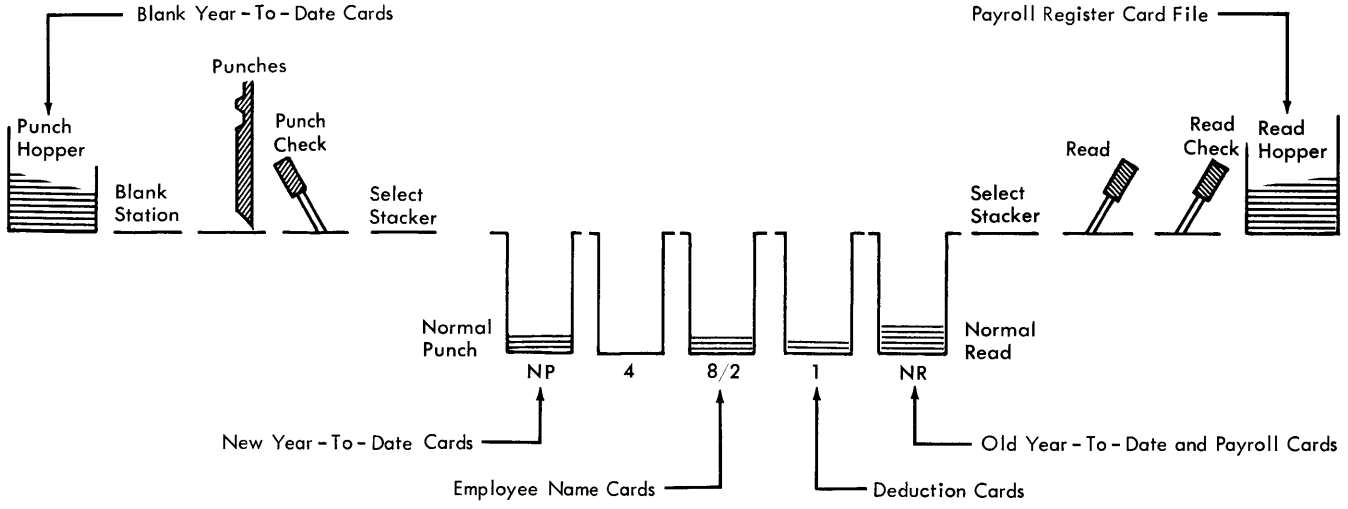


Figure 50. Card Stacking for Problem 7

year-to-date, deduction, and payroll), processes the data as required for each type, and prints the data for each man and department as specified in Figure 49.

The problem involves three calculations:

1. Exemption amount.
   $13.00 × number of dependents = exemption amount
2. Federal tax
   Current gross − exemption amount = taxable earnings
   Taxable earnings × 18% = federal tax
   Note: Not to be calculated if the taxable earnings amount is negative.
3. FICA tax
   Current gross × 3⅛% = FICA tax
   Note: Not to be calculated after Year-To-Date FICA reaches maximum FICA ($150.00).

Although the multiply-divide feature is available, it is used only to calculate FICA. Exemption amount and federal tax are calculated by using other techniques (details are given in the block diagram, Figure 51).

Labels used in the problem and their meanings are listed in the order they appear on the block diagram. Labels followed by DCW* indicate constants with a word mark.

Label	*Meaning*
START	Start of program (origin 0200).
RDDATE	Read date card punched 10/13/62 bbbbb41D in columns 65-80. 10/13/62 is week ending date, bbbbb are blank columns, 41 is roll number, and D is card code.
HALT1	Location of the HALT instruction when date card is missing.
HEAD	Location of the instruction for moving date information from date card to constant area.

Label		*Meaning*
DATE	DCW*	Storage area used for storing week-ending date and roll number (41).
RDNAME		Location of the instruction for reading first name card.
HALT2		Location of the HALT instruction when first name card is missing.
SEL2		Location of the instruction for selecting first name card.
EXEMPT	DCW*	Storage area used for calculating exemption amount (13 × TC). The amount is calculated by adding the tax code (punched in column 50) three times, and once more shifted one position to the left. Example:

Manual Method		*Machine Method*
TC (number of dependents)	5	5
Exemption amount for each dependent	× 13	5
Exemption amount	65	5 65

ONE	DCW*	Character 1 used for counting employees.
EMPCTR	DCW*	Storage area used for accumulating employee count by department.
MAJOR	DCW*	Hold area for storing department number for compare purposes.
MINOR	DCW*	Hold area for storing man number for compare purposes.
LASTCD		Last card test (sense switch A ON). Start of main program.
EOJ		Start of end-of-job routine.
TESTNA		Location of the instruction for testing name card after a minor control break.
HALT3		Location of the HALT instruction when first card of a man group is not a name card.

Label		*Meaning*
SELNAM		Location of the instruction for selecting name card when it is the first card of a man group.
YTDATE		Start of year-to-date card routine.
YDGROS	DCW*	Storage area for loading year-to-date gross.
YDFDTX	DCW*	Storage area for loading year-to-date federal tax.
YDFICA	DCW*	Storage area for loading year-to-date social security tax. FICA stands for Federal Insurance Contribution Act. It was originally known as OASI (Old Age Survivor's Insurance).
DEDUCT		Start of deduction card routine.
HALT4		Location of the HALT instruction when a card code other than -, &, 1, or blank is read in column 80.
DEGROS	DCW*	Storage area for accumulating department gross pay.
DEPTFT	DCW*	Storage area for accumulating department federal tax.
DEPTFIC	DCW*	Storage area for accumulating department FICA.
DEPDED	DCW*	Storage area for accumulating department deductions.
DEPTNP	DCW*	Storage area for accumulating department net pay.
MASK1	DCW*	Edit control word bb,bb0.bb (b indicates blanks).
MASK2	DCW*	Edit control word bb0.bb (b indicates blanks).
SW1		Internal switch that is automatically turned ON during end-of-job routine.
DEPTOT		Start of department total print routine. This occurs for each major control break, and end of job.
ATTHRS	DCW*	Storage area for accumulating department attendance hours.
CHAN1		Location of the instruction to SKIP TO CHANNEL 1 for all overflow sheets, and the beginning of each department.
SW2		Internal switch that is automatically turned ON during end-of-job routine.
FINSTP		Final HALT instruction when job is completed.
CHAN2		Start of routine for the first card (name) of a minor or major group. It is executed after minor or major totals are printed (as a result of a control change in man number or department number).
PAYROL		Start of the weekly payroll card routine.
AMPERS	DCW*	Ampersand (&) character for punching in new year-to-date card.
TAXEAR	DCW*	Storage area for computing taxable earnings (current earnings minus exemption amount).

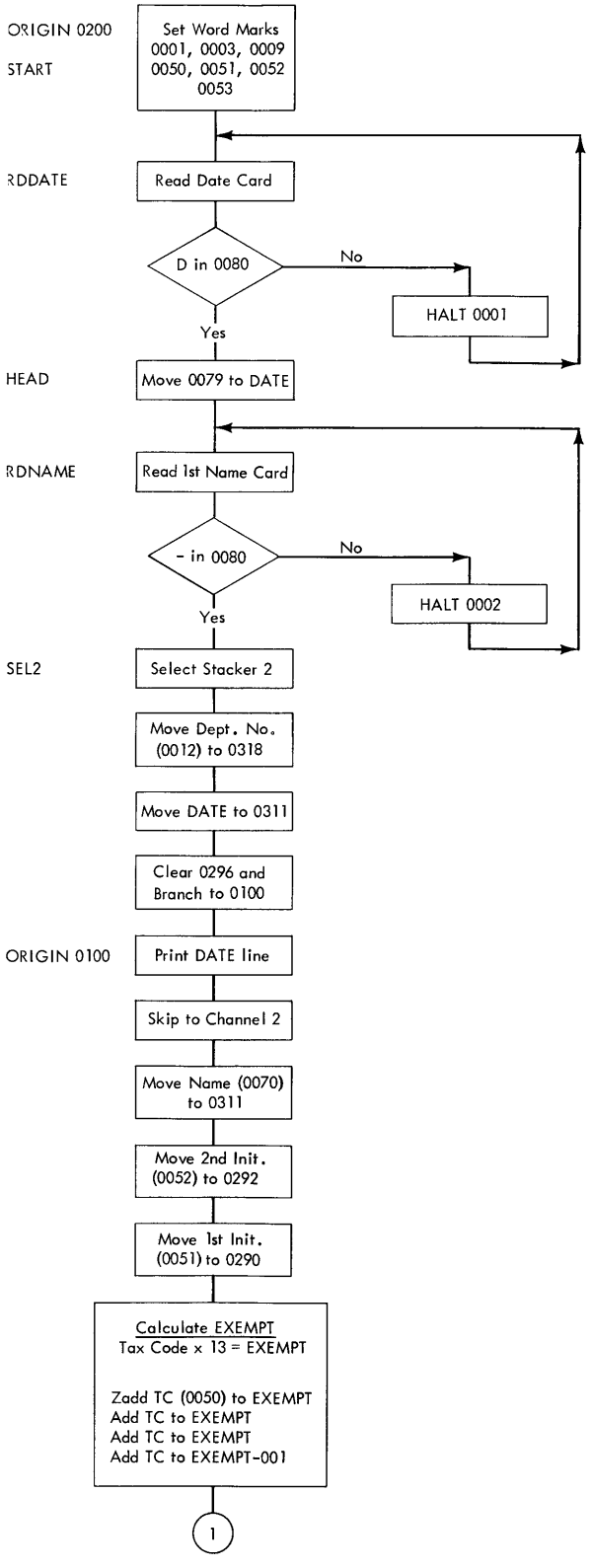


Figure 51. Block Diagram for Problem 7 (Part 1 of 5)

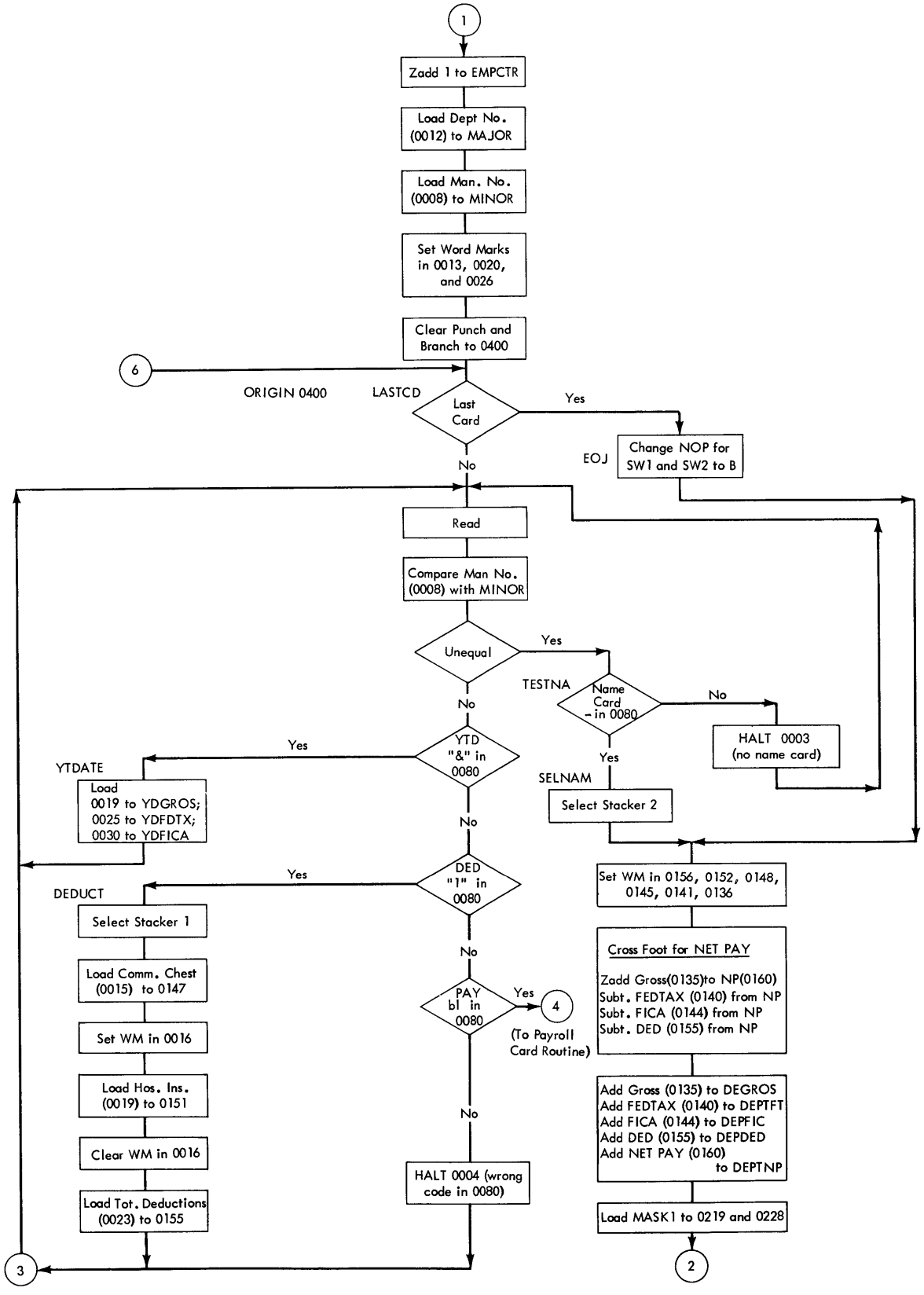


Figure 51. Block Diagram for Problem 7 (Part 2 of 5)

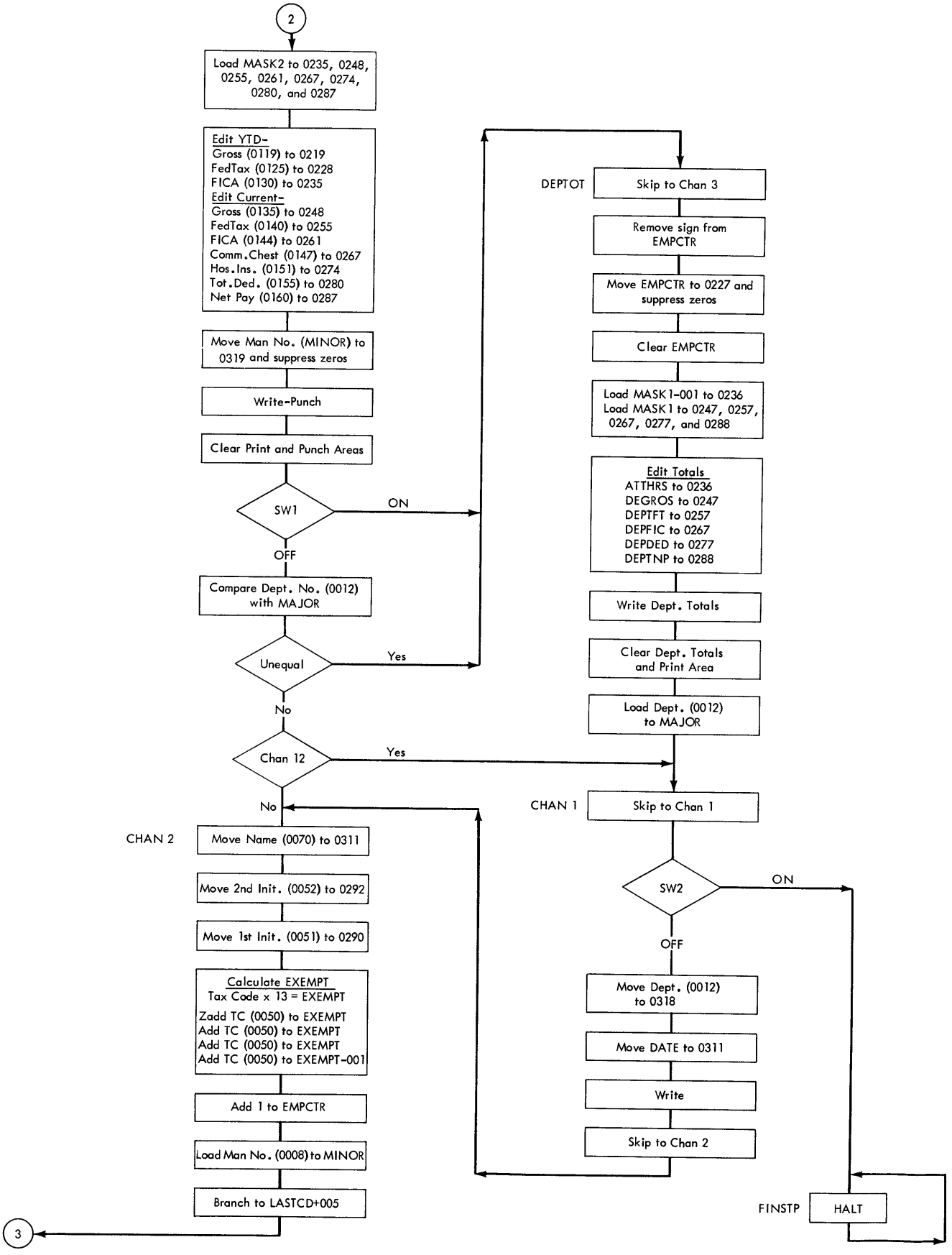


Figure 51. Block Diagram for Problem 7 (Part 3 of 5)

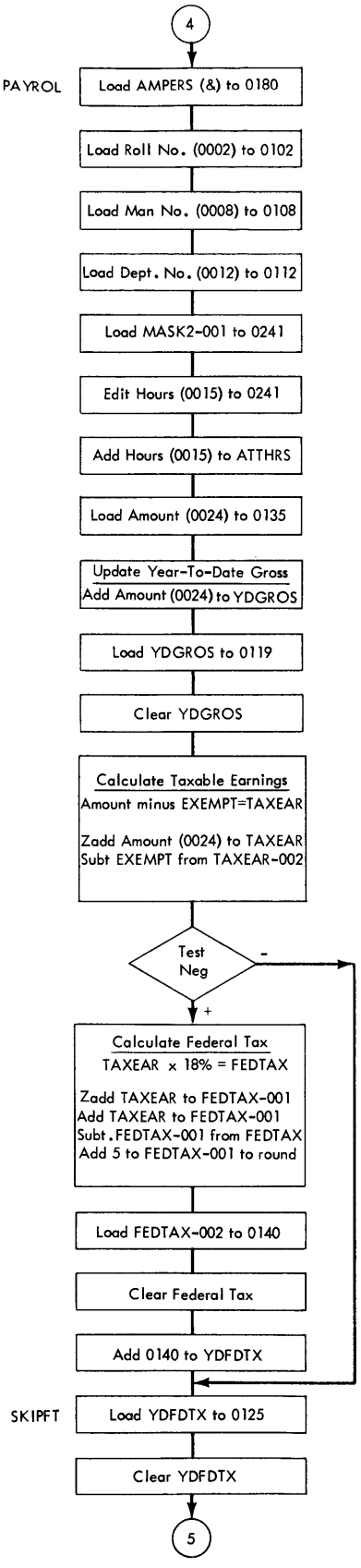


Figure 51. Block Diagram for Problem 7 (Part 4 of 5)

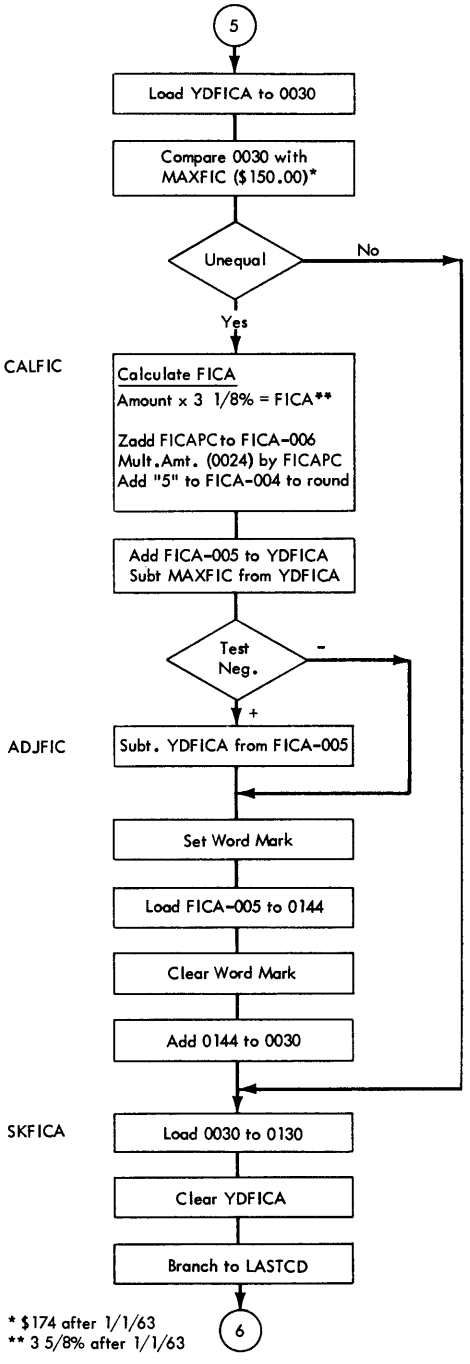


Figure 51. Block Diagram for Problem 7 (Part 5 of 5)

Label		*Meaning*
FEDTAX	DCW*	Storage area used for computing federal (withholding) tax. Normally, the federal tax is computed by multiplying taxable earnings by 18%. The method used in this example accomplishes the same result. The taxable earnings are added twice in this area (shifted one position to the left) then 10% of the result is subtracted. Example:

	Manual Method	*Machine Method*
Taxable earnings	1 20.59	12.0590 (10%)
	× .18	12.0590 (10%)
	9 64 72	24.1180 (20%)
	12 05 9	−2.4118 (− 2%)
	21.70 62	21.7062 (=18%)

FIVE	DCW*	Character 5 used for rounding federal tax and FICA.
SKIPFT		Start of routine after computing or skipping federal tax.
MAXFIC	DCW*	Storage area for holding maximum social security tax ($150.00).
CALFIC		Start of routine for calculating FICA.
FICA	DCW*	Storage area for calculating FICA (current gross × 3⅛% = FICA).
FICAPC	DCW*	Storage area for holding FICA percentage factor 3125. (Decimal equivalent of 3⅛% is .03125.)
ADJFIC		Instruction for adjusting FICA when the sum of year-to-date FICA and current FICA exceeds $150.00.
SKFICA		Start of routine after computing or skipping FICA.
ON	DCW*	Character B for turning switches 1 and 2 ON during end-of-job routine.

Figure 52 shows the symbolic program for *Problem 7*. It includes two operation codes not used in previous problems:

Instruction	*Mnemonic*	*Actual*
1. Branch if Character Equal	B	B
2. Branch if Word Mark and/or Zone	BWZ	V

The BRANCH IF CHARACTER EQUAL instructions in Figure 52 test the five different types of card codes (-, &, 1, blank, and D) in column 80. The branch to the A-address occurs only if the card type code in the location specified by the B-address is equal to the d-modifier in the instruction.

The BRANCH IF WORD MARK AND/OR ZONE instruction (BWZ) tests the results of an accumulator after adding and subtracting. The instruction can be used to test for either a plus or minus sign in the units position of the accumulator. The d-modifier is B (12-zone) to test plus and K (11-zone) to test minus. The branch to the A-address occurs when the sign in the units position of the accumulator (specified by the B-address) is equal to the sign B or K (specified by the d-modifier).

A NOP instruction is used to place a word mark immediately after the last character in the two initialization routines. It is also used in the main routine in conjunction with an A-address (Nxxx) to set up two internal switches. When the operation code is N, the switches are OFF. Therefore, no branch occurs. When the operation code is changed to B during the end-of-job routine, the switches are ON. Therefore, an unconditional branch occurs to the location specified by the A-address.

Problem 8. Payroll Check and Earnings Statement

This problem involves preparing a weekly payroll check and earnings statement, and uses three different types of cards:

Type of Card	*Card Type Code*
1. Date and starting check number	D
2. New year-to-date	&
3. Employee name	-

The date (10/13/62) and starting check-number (12490) card precedes the file. The file was originally two files: the new year-to-date file punched during *Problem 7*, and the employee name file selected in stacker 2. The two files were merged in the order listed in department and man number sequence.

In addition to the payroll check and earnings statement, a check duplicate is to be punched for each check written. This card will be used to prepare check registers, and for check reconciliation.

Figure 53 shows the formats for the payroll check, earnings statement, the three types of input cards, and the one output card.

Payroll Check and Earnings Statement Specifications

The payroll check and earnings statement will be prepared on a preprinted form (as shown in Figure 53). The form is 12 inches wide (left to right) and 3 inches long (top to bottom). The three lines of printing for the first form are specified by holes in the carriage tape that are punched as follows:

First line = channel 1, line 4.
Second line = channel 2, line 9.
Third line = channel 3, line 14.

This marking for one three-inch form (18 lines) is repeated to control seven forms in one revolution of the tape. The tape is cut at line 126 (18 lines per form ×7).

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL REGISTER

Page No. 01 of 12

Programmed by _____ Date _____

Identification PROB 7

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
3 5	6 7	8 13	14 16	17	23		27	28	34		38	39	40 55
010			CTL	33									
020		*THIS	RO	UTINE SETS				UP HEADING				D	DATE LINE FOR
030		*FI	RST	FORM AND S				ELECT FIRST					NAME CARD
040			ORG	02000									INITIALIZING
050		START	SW	00001				00037					ROUTINE TO SETUP
060			SW	00009				00050					HEADING LINE
070			SW	00051				00052					FIRST HALT MEANS
080			SW	00053									NO DATE CARD
090		RDDATE	R										SECOND HALT
100			B	HEAD				00800				D	MEANS FIRST FILE
110		HALT1	H	00000				00001					CARD IS NOT A
120			B	RDDATE									NAME CARD
130		HEAD	MCW	00079				DATE					STORE DATE & ROLL#
140		RDNAME	R										
150			B	SEL2				00800				-	TEST FOR NAME
160		HALT2	H	00000				00002					CARD
170			B	RDNAME									
180		SEL2	SS									2	SELECT NAME CARD
190			MCS	00012				03168					MOVE DEPT AND
200			MCW	DATE				03111					DATE - - CLEAR ROUT
201			CS	01000				02966					INE AND BRANCH
202			NOP										TO 01000

Figure 52. Symbolic Program for Problem 7 (Part 1 of 12)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL REGISTER

Page No. 02 of 12

Programmed by _____ Date _____

Identification PROB 7

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
3 5	6 7	8 13	14 16	17	23		27	28	34		38	39	40 55
010		*THIS	RO	UTINE PRINT				S HEADING				L	INE ON FIRST FORM
020		*SKIP	ST	O CHANNEL 2				LOADS NAME					FROM FIRST CARD
030		*TO PR	INT	AND CALCU				LATES EXEMPT					ION AMOUNT
040			ORG	01000									
050			W										WRITE DATE LINE
060			CC									2	SKIP TO CHAN 2
070			MCW	00700				03111					MOVE LAST NAME
080			MCW	00052				02922					MOVE 2ND INITIAL
090			MCW	00051				02900					MOVE 1ST INITIAL
100			ZA	00050				EXEMPT					CALCULATE
110			A	00050				EXEMPT					EXEMPTION AMOUNT
120			A	00050				EXEMPT					TAX CODE
130			A	00050				EXEMPT	-	0001			TIMES 13
140			ZA	ONE				EMPCTR					ADD 1 TO EMPCTR
150			LCA	00012				MAJOR					LD DEPT TO MAJOR
160			LCA	00008				MINOR					LD MAJNO TO MINOR
170			SW	00013				00020					SET WORD MARKS
180			SW	00026									
190			CS	04000				01999					CLEAR ROUTINE
200			NOP										AND BR TO 04000

Figure 52. Symbolic Program for Problem 7 (Part 2 of 12)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL REGISTER

Page No. 03 of 12

Programmed by _____ Date _____

Identification PROB7

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010		*THIS	IS	THE START				OF THE		MAIN			ROUTINE
020			ORG	0400									
030		LASTCD	B	EOJ								A	BR IF LAST CARD
040			R										READ A CARD
050			C	0008				MINOR					COMPARE MAN NO.
060			B	TESTNA								/	BR IF UNEQUAL
070			B	YTDATE				0080				&	TEST YTD & BR
080			B	DEDUCT				0080				1	TEST DEDUCT & BR
090			B	PAYROL				0080					TEST PAYROL & BR
100		HALT4	H	0000				0004					WRONG CODE IN 80
110			B	LASTCD	+	005							BR TO READ
120		TESTNA	B	SELNAM				0080				-	TEST NAME
130		HALT3	H	0000				0003					NO NAME CARD
140			B	LASTCD	+	005							BR TO READ
150		SELNAM	SS									2	SELECT NAME CARD
160			SW	0156				0152					SET WORD MARKS
170			SW	0148				0145					IN PUNCH AREA BE
180			SW	0141				0136					FORE NET PAY CALC
190			ZA	0135				0160					ZERO ADD CURGROS
200			S	0140				0160					SUBT CURFEDTAX

Figure 52. Symbolic Program for Problem 7 (Part 3 of 12)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL REGISTER

Page No. 04 of 12

Programmed by _____ Date _____

Identification PROB7

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010			S	0144				0160					SUBT CURRENTFICA
020			S	0155				0160					SUBT CUR DEDUCTS
030			A	0135				DEGROS					ADD CURR GROSS,
040			A	0140				DEPTFT					FEDTAX, FICA, DE-
050			A	0144				DEPFIC					DUCTIONS AND
060			A	0155				DEPDED					NET PAY TO DEPT.
070			A	0160				DEPTNP					ACCUMULATORS
080			LCA	MASK1				0219					LD EDIT CTL WORD
090			LCA	MASK1				0228					FOR YTD GROSS & FIT
100			LCA	MASK2				0235					LD EDIT CTL WORD
110			LCA	MASK2				0248					FOR YTD, FICA AND
120			LCA	MASK2				0255					CURRENT GROSS,
130			LCA	MASK2				0261					FEDTAX, FICA, COM.
140			LCA	MASK2				0267					CHEST, HOS., INS.,
150			LCA	MASK2				0274					TOTAL DEDUCTIONS
160			LCA	MASK2				0280					AND NET PAY
170			LCA	MASK2				0287					
180			MCE	0119				0219					EDIT YTD GROSS,
190			MCE	0125				0228					YTD FEDTAX AND
200			MCE	0130				0235					YTD FICA

Figure 52. Symbolic Program for Problem 7 (Part 4 of 12)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL REGISTER

Page No. 05 of 12

Programmed by _____ Date _____

Identification PROB7

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010			MCE	0135				0248					EDIT CURRENT
020			MCE	0140				0255					GROSS, FEDTAX,
030			MCE	0144				0261					FICA, COMM. CHEST,
040			MCE	0147				0267					HOS. INS. TOTAL
050			MCE	0151				0274					DEDUCTIONS AND
060			MCE	0155				0280					NET PAY
070			MCE	0160				0287					
080			MCS	MINOR				0319					M. MANNO. SUPZEROS
090			WP										WRITE-PUNCH
100			CS	0332									CLEAR PRINT
110			CS										AND PUNCH
120			CS										AREAS
130		SW1	NOP	DEPTOT									NOP OFF BRANCHON
140			C	0012				MAJOR					COMPARE DEPT. NO.
150			B	DEPTOT								/	BRANCH UNEQUAL
160			B	CHAN1								@	TEST CHAN12 & BR
170		CHAN2	MCW	0070				0311					MOVE NAME
180			MCW	0052				0292					MOVE 2ND INITIAL
190			MCW	0051				0290					MOVE 1ST INITIAL
200			ZA	0050				EXEMPT					CALCULATE

Figure 52. Symbolic Program for Problem 7 (Part 5 of 12)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL REGISTER

Page No. 06 of 12

Programmed by _____ Date _____

Identification PROB7

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010			A	0050				EXEMPT					EXEMPTION AMOUNT
020			A	0050				EXEMPT					TAX CODE
030			A	0050				EXEMPT	-	001			TIMES 13
040			A	ONE				EMPCTR					ADD 1 TO COUNTER
050			LCA	0008				MINOR					LOAD MANNO TOMIN
060			B	LASTCD	+	005							BR TO READ
070		* YEAR-TO-DATE CARD						ROUTINE					
080		YTDATE	LCA	0019				YDGROS					LOAD YEAR-TO-
090			LCA	0025				YDFDTX					DATE GROSS,
100			LCA	0030				YDFICA					FEDTAX AND FICA
110			B	LASTCD	+	005							BRANCH TO READ
120		* DEDUCTION CARD ROUTINE											
130		DEDUCT	SS									1	SELECT DEDUCT CD
140			LCA	0015				0147					LOAD COMM. CHEST
150			SW	0016									SET WM FOR HOS IN
160			LCA	0019				0151					LOAD HOS. INS.
170			CW	0016									CLEAR WM HOS. INS
180			LCA	0023				0155					LOAD TOT. DEDUCT
190			B	LASTCD	+	005							BRANCH TO READ
200													

Figure 52. Symbolic Program for Problem 7 (Part 6 of 12)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL REGISTER

Page No. 07 of 12

Programmed by _____ Date _____

Identification PROB7

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND	d	COMMENTS
010		*THIS	RO	UTINE CALC				ULATES CURRE					NT FEDERAL AND
020		*FICA	TA	XES AND UP				DATES YEAR-T					O-DATE GROSS, FED-
030		*ERAL	AND	FICA TAXE				S--CURRENT					FEDERAL TAX IS
040		*COMP	UTE	D ON TAXAB				LE EARNINGS,					AND CURRENT FICA
050		*TAX	IS	COMPUTED ON				LY FOR EMPL				0	YEES WHOSE YEAR-
060		*TO-DATE		FICA IS LE				SS THAN 81				0	.00--ALSO CALCU
070		*LATES	F	INAL FICA	T	AX							
080		*											
090		PAYROLL	LCA	AMPERS				0180					LOAD CARD CODE &
100			LCA	0002				0102					LOAD ROLL NUMBER
110			LCA	0006				0108					MAN NO., DEPT. NO.
120			LCA	0012				0112					TO PUNCH AREA
130			LCA	MASK2	-	0001		0241					LOAD EDIT WORD
140			MCE	0015				0241					EDIT HOURS
150			A	0015				ATTHRS					ADD HRS TO ATTHR
160			LCA	0024				0135					LOAD AMOUNT
170			A	0024				YDGROS					UPDATE YDGROS
180			LCA	YDGROS				0119					LOAD NEW YDGROS
190			S	YDGROS									CLEAR YDGROS
200			ZA	0024				TAXEAR					ADD AMT TOTAXEAR

Figure 52. Symbolic Program for Problem 7 (Part 7 of 12)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL REGISTER

Page No. 08 of 12

Programmed by _____ Date _____

Identification PROB7

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND	d	COMMENTS
010			S	EXEMPT				TAXEAR	-	0062			SUBTRACT EXEMPT
020			BWZ	SKIPFT				TAXEAR	-	0002		K	BR IF TAXEAR IS-
030			ZA	TAXEAR				FEDTAX	-	0001			COMPUTE FEDTAX
040			A	TAXEAR				FEDTAX	-	0001			
050			S	FEDTAX	-	0001		FEDTAX					
060			A	FIVE				FEDTAX	-	0001			ROUND FEDTAX
070			LCA	FEDTAX	-	0002		0140					LOAD FEDTAX
080			S	FEDTAX									CLEAR FEDTAX
090			A	0140				YDFDTX					UPDATE YDFDTX
100		SKIPFT	LCA	YDFDTX				0125					LOAD NEW YDFDTX
110			S	YDFDTX									CLEAR YDFDTX
120			LCA	YDFICA				0030					LOAD YDFICA
130			C	0030				MAXFIC					COMPARE WITH 150
140			B	CALFIC								/	UNEQ COMPUTE FICA
150			B	SKFICA									EQUAL SKIP FICA
160		CALFIC	ZA	FICAPC				FICA	-	0006			LOAD MPR 3125
170			M	0024				FICA					CALCULATE FICA
180			A	FIVE				FICA	-	0004			ROUND FICA
190			A	FICA	-	0005		YDFICA					UPDATE YDFICA
200			S	MAXFIC				YDFICA					SUBTRACT 15000

Figure 52. Symbolic Program for Problem 7 (Part 8 of 12)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL REGISTER

Page No. 09 of 12

Programmed by _____ Date _____

Identification PROB7

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010			BWZ	ADJFIC	+	007		YDFICA				K	BR IF YDFICA IS-
020		ADJFIC	S	YDFICA				FICA	-	005			REDUCE CURR FICA
030			SW	FICA	-	008							SET WM FOR FICA
040			LCA	FICA	-	005		0144					LD FICA TO PUNCH
050			CW	FICA	-	008							CLEAR WM FICA
060			A	0144				0030					UPDATE YDFICA
070		SKFICA	LCA	0030				0130					LOAD NEW YDFICA
080			S	YDFICA									CLEAR FICA
090			B	LASTCD									BRANCH TO LASTCD
100		* THIS	RO	UTINE PRINTS DEPARTMENT TOTALS WHEN									
110		* COMPARE DEPT IS UNEQUAL											
120		DEPTOT	CC									3	SKIP TO CHANNEL 3
130			MZ	0099				EMPCTR					REMOVE PLUS SIGN
140			MCS	EMPCTR				0227					MOVE EMPCTR & SUPZ
150			S	EMPCTR									CLEAR EMPCTR
160			LCA	MASK1	-	001		0236					LOAD EDIT
170			LCA	MASK1				0247					CONTROL WORD
180			LCA	MASK1				0257					, 0
190			LCA	MASK1				0267					
200			LCA	MASK1				0277					
201			LCA	MASK1				0288					

Figure 52. Symbolic Program for Problem 7 (Part 9 of 12)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL REGISTER

Page No. 10 of 12

Programmed by _____ Date _____

Identification PROB7

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010			MCE	ATTHRS				0236					EDIT ATTHRS,
020			MCE	DEGROS				0247					DEPT. GROSS, DEPT.
030			MCE	DEPTFT				0257					FEDTAX, DEPT. FICA
040			MCE	DEPFIC				0267					DEPT. DEDUCTS, AND
050			MCE	DEPDED				0277					DEPT. NET PAY
060			MCE	DEPTNP				0288					
070			W										WRITE DEPT. TOTAL
080			S	ATTHRS									CLEAR
090			S	DEGROS									DEPARTMENT
100			S	DEPTFT									TOTALS
110			S	DEPFIC									
120			S	DEPDED									
130			S	DEPTNP									
140			CS	0332									CLEAR PRINT AREA
150			CS										
160			LCA	0012				MAJOR					STORE DEPT. NO.
170		CHAN1	CC									1	SKIP TO CHANNEL 1
180		SW2	NOP	FINSTP									NOP OFF BRANCH ON
190			MCW	0012				0318					MOVE DEPT. NO.
200			MCW	DATE				0311					MOVE DATE
201			W										WRITE DATE LINE
202			CC	CHAN2								2	SKIP TO CHAN2 & BR

Figure 52. Symbolic Program for Problem 7 (Part 10 of 12)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL REGISTER

Page No. 11 of 12

Programmed by _____ Date _____

Identification PROB7

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010		* END-	OF-	JOB ROUTINE									
020		EOJ	MCW	ON				SW1					CHANGE NOP FOR
030			MCW	ON				SW2					SW1 AND SW2 TO B
040			B	SELNAM	+	002							
050		FINSTP	H	FINSTP									END-OF-JOB HALT
060		*											
070		* CO	NS	TANTS									
080	15	DATE	DCW	*									
090	03	EXEMPT	DCW	*									
100	03	EMPCTR	DCW	*									
110	04	MAJOR	DCW	*									
120	06	MINOR	DCW	*									
130	07	YDGROS	DCW	*									
140	06	YDFDTX	DCW	*									
150	05	YDFICA	DCW	*									
160	07	DEGROS	DCW	*									
170	06	DEPTFT	DCW	*									
180	06	DEPFIC	DCW	*									
190	06	DEPDED	DCW	*									
200	07	DEPTNP	DCW	*									

Figure 52. Symbolic Program for Problem 7 (Part 11 of 12)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL REGISTER

Page No. 12 of 12

Programmed by _____ Date _____

Identification PROB7

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010	09	MASK1	DCW	*		,0		0.					
020	06	MASK2	DCW	*		0	.						
030	05	ATTHRS	DCW	*									
040	05	TAXEAR	DCW	*									
050	07	FEDTAX	DCW	*									
060	05	MAXFIC	DCW	*		15000							
070	04	FICAPC	DCW	*		3125							
080	10	FICA	DCW	*									
090	01	FIVE	DCW	*		5							
100	01	ONE	DCW	*		1							
110	01	AMPERS	DCW	*		&							
120	01	ON	DCW	*		B							
130			END	START									
140													
150													
160													
170													
180													
190													
200													

Figure 52. Symbolic Program for Problem 7 (Part 12 of 12)

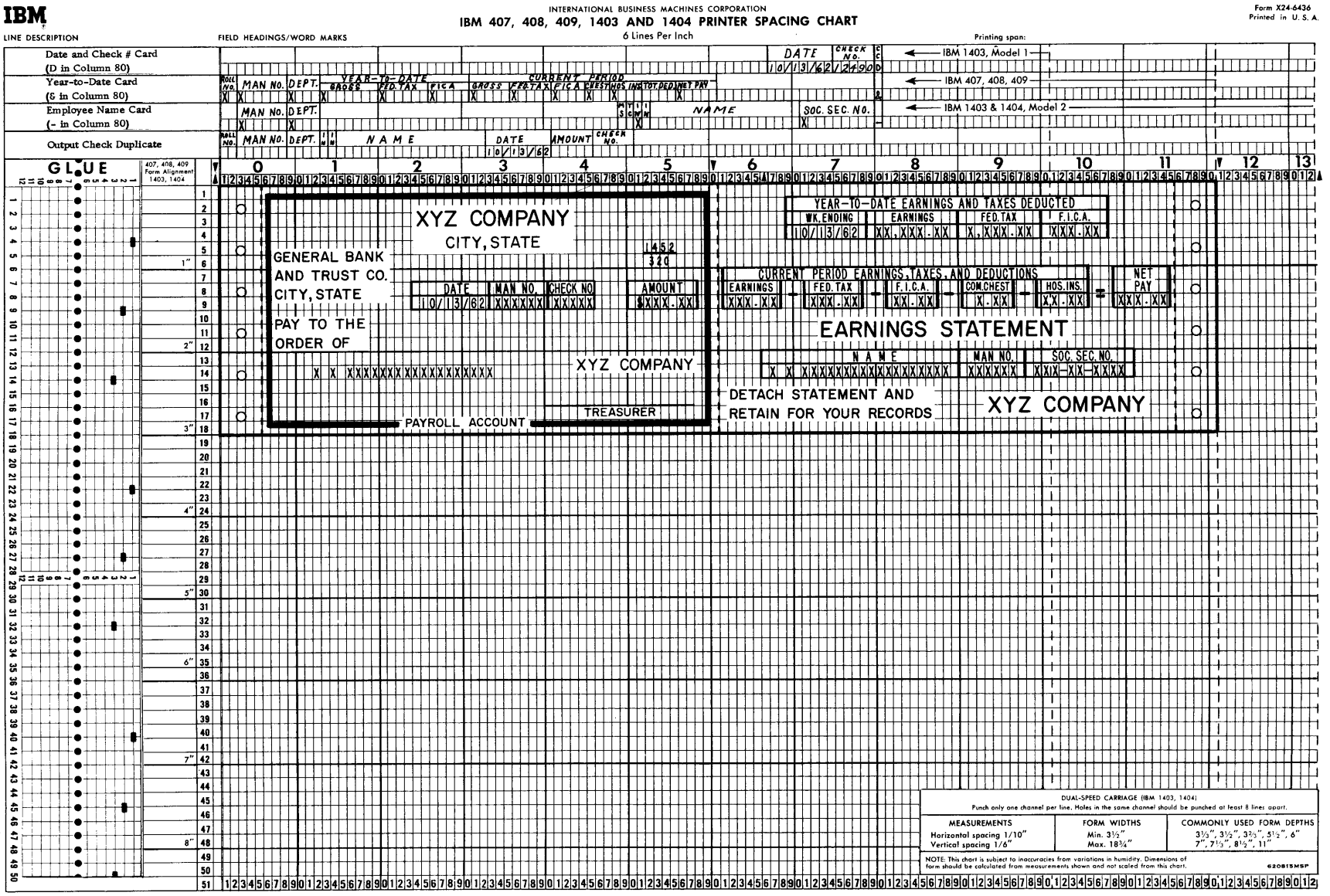


Figure 53. Printer Specifications for Problem 8

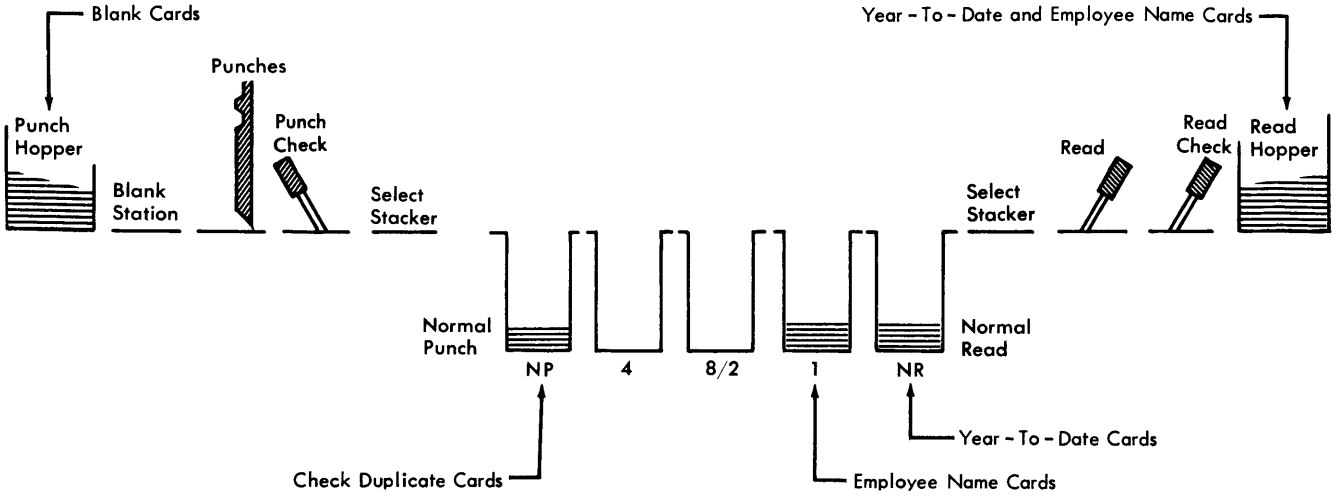


Figure 54. Card Stacking for Problem 8

Program Halts

The program must include instructions to stop the machine under the following conditions:

Initialization Routine

Halt 1 = No date card

Main Routine

Halt 2 = First card of a man group after a control break is not a year-to-date card

Halt 3 = Second card of a man group is not a name card

Final Halt = End of job

Earnings Statement Option

Although usually both the payroll check and earnings statement are prepared, occasionally the data on the earnings statement is not required. In such cases, eliminate printing on the statement by use of sense switch B for the first line, C for the second line, and D for the third line.

Card Stacking

The program must provide for stacking cards as shown in Figure 54.

Block Diagram

The block diagram is shown in Figure 55. The program originates at location 0400. During initialization, word marks are set and a test is made for the presence of the date and starting check number card. If absent, the program HALT1 is executed, and the machine stops. If present, the date and check number are stored in hold areas.

The main routine originates in the location labeled LASTCD. The routine tests for the two types of cards, stops the machine if either card is missing, and processes the data as required for each check and earnings statement if both cards are present.

You can eliminate printing on the statement, if you desire, by setting sense switches B, C, and D ON before you start the operation. Sense switches are not standard but may be installed as special features. There are six switches labeled B, C, D, E, F, and G. They are very useful for eliminating routines in the main routine, or branching to subroutines at the option of the operator.

Labels used in the problem and their meanings are listed in the order they appear on the block diagram. Labels followed by DCW* indicate constants with a word mark.

Label		*Meaning*
START		Start of program (origin 0400).
RDDATE		Read date and starting check number card. Date (10/13/62) is punched in columns 67-74 and check number (12490) is punched in columns 75-79. The card type code (D) is punched in column 80.
HALT1		Location of the HALT instruction when date card is missing.
STORDA		Location of the instruction for storing check number and date into hold areas.
CHEKNO	DCW*	Storage area for storing starting check number.
DATE	DCW*	Storage area for storing date.
LASTCD		Last card test (sense switch A ON). Start of main routine.
ENDJOB		Start of end-of-job routine.
COMPAR		Location of the instruction to compare man number punched in the card with man number in storage.
CONTRL	DCW*	Hold area for storing man number for compare purposes.
CHAN1		Location of the instruction to SKIP TO CHANNEL 1 for the first card of each man group.

Label		*Meaning*
HALT2		Location of the HALT instruction when the year-to-date card is not the first card of a man group.
MASK4	DCW*	Edit control word b&b&bbbbbbbbbb bbbbbbbb for name (b indicates blanks).
MASK5	DCW*	Edit control word bbb-bb-bbbb for social security number (b indicates blanks).
WRIPUN		Location of the instruction for printing the third line and punching a check duplicate.
ONE	DCW*	Character 1 for increasing check numbers.
YRDATE		Start of year-to-date card routine.
MASK1	DCW*	Edit control word bb, bb0.bb (b indicates blanks).
MASK2	DCW*	Edit control word bb0.bb (b indicates blanks).
CHAN2		Location of the instruction to SKIP TO CHANNEL 2 for printing the second line. The skip occurs after printing the first line.
MASK3	DCW*	Edit control word bb$0.bb for check amount (b indicates blanks).
CHAN3		Location of the instruction to SKIP TO CHANNEL 3 for printing the third line. The skip occurs after printing the second line.
READ		Location of the instruction to READ AND TEST FOR NAME CARD (second card of a man group).
HALT3		Location of the HALT instruction when the name card is missing.

The symbolic program for *Problem 8* is shown in Figure 56.

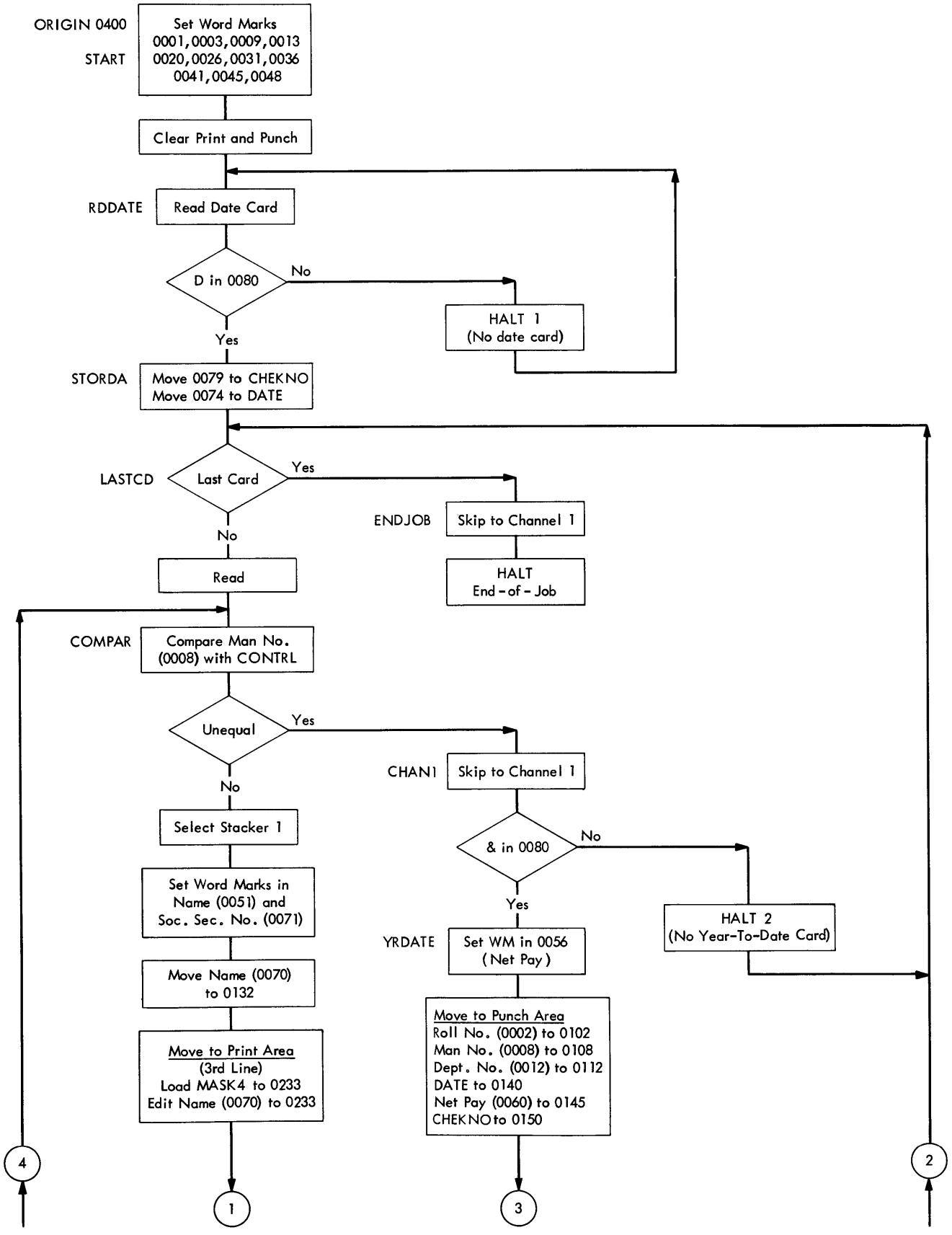


Figure 55. Block Diagram for Problem 8 (Part 1 of 2)

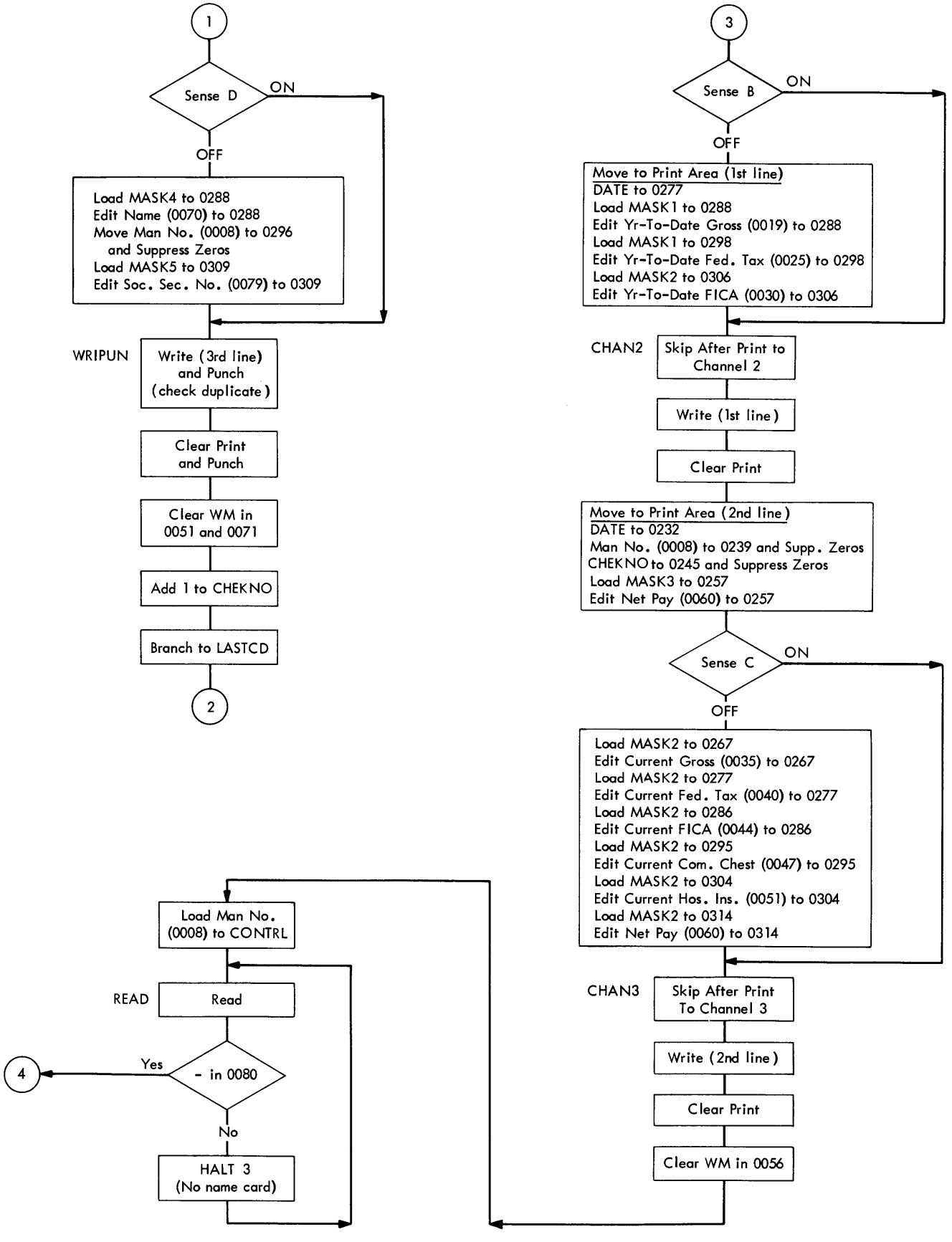


Figure 55. Block Diagram for Problem 8 (Part 2 of 2)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL CHECK AND EARNINGS STATEMENT CODING SHEET

Page No. 01 of 5

Programmed by _____ Date _____

Identification PROB F

LINE 3-5	COUNT 6-7	LABEL 8-13	OPERATION 14-16	(A) OPERAND ADDRESS 17-23	(A) OPERAND ±	(A) OPERAND CHAR. ADJ.	IND 27	(B) OPERAND ADDRESS 28-34	(B) OPERAND ±	(B) OPERAND CHAR. ADJ.	IND 38	d 39	COMMENTS 40-55
010			CTL	33									
020		*PAYR	OLL	CHECK AND				EARNINGS ST					ATEMENT
030		*PRINT	ING ON EARNI					NGS STATEMENT					ELIMINATED WHEN
040		*SENS	E SWITCHES B,					C, AND D ARE					TURNED ON
050			ORG	0400									
060		START	SW	0001				0003					SET WORD
070			SW	0009				0013					MARKS
080			SW	0020				0026					
090			SW	0031				0036					
100			SW	0041				0045					
110			SW	0048									
120			CS	0332									CLEAR PRINT AND
130			CS										PUNCH AREAS
140			CS										
150		RDDATE	R										READ DATE CARD
160			B	STORDA				0080				D	TEST FOR DATE CD
170		HALT1	H	0000				0001					HALT NO DATE CD
180			B	RDDATE									
190		STORDA	MCW	0079				CHEKNO					STORE CHECK NO.
200			MCW										STORE DATE

Figure 56. Symbolic Program for Problem 8 (Part 1 of 5)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL CHECK AND EARNINGS STATEMENT CODING SHEET

Page No. 02 of 5

Programmed by _____ Date _____

Identification PROB F

LINE 3-5	COUNT 6-7	LABEL 8-13	OPERATION 14-16	(A) OPERAND ADDRESS 17-23	(A) OPERAND ±	(A) OPERAND CHAR. ADJ.	IND 27	(B) OPERAND ADDRESS 28-34	(B) OPERAND ±	(B) OPERAND CHAR. ADJ.	IND 38	d 39	COMMENTS 40-55
010		LASTCD	B	ENDJOB								A	BR IF LAST CARD
020			R										READ A CARD
030		COMPARE	C	0008				CONTRL					COMPARE MAIN NO.
040			B	CHAN1								/	UNEQ. BR TO CHAN1
050			SS									1	SELECT NAME CARD
060			SW	0051				0071					SET WM NAME SSNO
070			MCW	0070				0132					MOVE NAME TO PCH
080			LCA	MASK4				0233					LOAD EDIT WORD
090			MCE	0070				0233					EDIT NAME
100			B	WRIPUN								D	BR SENSE D ON
110			LCA	MASK4				0288					LOAD EDIT WORD
120			MCE	0070				0288					EDIT NAME
130			MCS	0008				0296					MOVE MANNO & SUPPZ
140			LCA	MASK5				0309					LOAD EDIT WORD
150			MCE	0079				0309					EDIT SOC. SEC. NO.
160		WRIPUN	WP										WR 3RD LINE PCH CD
170			CS	0332									CLEAR PRINT AND
180			CS										PUNCH AREAS
190			CS										
200			CW	0051				0071					CLEAR WORD MARKS

Figure 56. Symbolic Program for Problem 8 (Part 2 of 5)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL CHECK AND EARNINGS STATEMENT CODING SHEET

Page No. 03 of 5

Programmed by _____ Date _____ Identification PROB 5

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND	d	COMMENTS
010			A	ONE				CHEKNO					INCREASE CHECKNO
020			B	LAST CD									BR TO LAST CARD
030		CHAN1	CC									1	SKIP TO CHANNEL 1
040			B	YRDATE				0080				&	TEST YR-TO-DATE
050		HALT2	H	0000				0002					HALT NO YR-TO-DA
060			B	LAST CD									BR TO LAST CARD
070		YRDATE	SW	0056									SET WM FOR NET P
080			MCW	0002				0102					MOVE ROLL NO.
090			MCW	0008				0108					MOVE MAN NO.
100			MCW	0012				0112					MOVE DEPT. NO.
110			MCW	DATE				0140					MOVE DATE
120			MCW	0060				0145					MOVE AMOUNT
130			MCW	CHEKNO				0150					MOVE CHECK NO.
140			B	CHAN2								B	BR SENSE B ON
150			MCW	DATE				0277					MOVE DATE
160			LCA	MASK1				0288					LOAD EDIT WORD
170			MCE	0019				0288					EDIT YTD GROSS
180			LCA	MASK1				0298					LOAD EDIT WORD
190			MCE	0025				0298					EDIT YTD FEDTAX
200			LCA	MASK2				0306					LOAD EDIT WORD
201			MCE	0030				0306					EDIT YTD FICA

Figure 56. Symbolic Program for Problem 8 (Part 3 of 5)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL CHECK AND EARNINGS STATEMENT CODING SHEET

Page No. 04 of 5

Programmed by _____ Date _____ Identification PROB 5

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND	d	COMMENTS
010		CHAN2	CC									B	SKIP TO CHANNEL 2
020			W										WRITE FIRST LINE
030			CS	0332									CLEAR PRINT AREA
040			CS										
050			MCW	DATE				0232					MOVE DATE
060			MCS	0008				0239					MOVE MANNO & SUPPZ
070			MCS	CHEKNO				0245					M CHEKNO & SUPPZ
080			LCA	MASK3				0257					LOAD EDIT WORD
090			MCE	0060				0257					EDIT AMOUNT
100			B	CHAN3								C	BR SENSE C ON
110			LCA	MASK2				0267					LOAD EDIT WORD
120			MCE	0035				0267					EDIT CURR GROSS
130			LCA	MASK2				0277					LOAD EDIT WORD
140			MCE	0040				0277					EDIT CURR FEDTAX
150			LCA	MASK2				0286					LOAD EDIT WORD
160			MCE	0044				0286					EDIT CURR FICA
170			LCA	MASK2				0295					LOAD EDIT WORD
180			MCE	0047				0295					EDIT CUR COMCHES
190			LCA	MASK2				0304					LOAD EDIT WORD
200			MCE	0051				0304					EDIT CUR HOS. INS

Figure 56. Symbolic Program for Problem 8 (Part 4 of 5)

Part 3. Introduction to Tape Records

As a result of a study made by a team of systems planners, the XYZ Company decided to convert their punched card master files to magnetic tape. Henceforth, the various reports and documents will be prepared with transaction cards punched during current periods, and master magnetic tape files.

In your particular area, three master card files are already converted to magnetic tape. They are:

1. Employee payroll master name file
2. Year-to-date earnings file
3. Payroll deduction file.

The files were converted with no change in record format, by use of a standard card-to-tape utility program supplied by IBM. The program can be tailored to fit a particular job by punching a few control cards to modify the program. Other utility programs are available, (such as tape-to-card, tape-to-printer, and multiple utility) that permit any one, two, or all three tape utility programs to occur concurrently. The mode of operation is under sense-switch control.

In addition to the utility programs mentioned, other standard programs are available, such as:

Input/Output Control System (IOCS). This system eliminates to a large extent the detailed programming required for input/output operations.

Tape Sorting and Merging (Sort 1 and Sort 2-Merge 2). These programs sort and/or merge tape records.

Report Program Generator (RPG). This standard generator makes it easier to prepare programs to write reports of variable format from data stored either in punched cards or in magnetic tape.

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM

FORM X24-1152
PRINTED IN U.S.A.

Program PAYROLL CHECK AND EARNINGS STATEMENT CODING SHEET

Page No. 05 of 5

Programmed by _____ Date _____

Identification PROB 8

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND.	d	COMMENTS
010			LCA	MASK2				03,14					LOAD EDIT WORD
020			MCE	0060				03,14					EDIT CUR NETPAY
030		CHAN3	CC									C	SKIP TO CHANNEL 3
040			W										WRITE SECND LINE
050			CS	0332									CLEAR PRINT AREA
060			CS										
070			CW	0056									CLEAR WM NETPAY
080			LCA	0008				CONTRL					STORE MAN NO.
090		READ	R										READ A CARD
100			B	COMPAR				0080				-	BR TO COMP IF-80
110		HALT3	H	0000				0003					HALT NO NAME
120			B	READ									
130		ENDJOB	CC									1	SKIP TO CHANNEL 1
140			H										END-OF-JOB HALT
150		*CON	NS	TAN	TS								
160	08	DATE	DCW	*		0000	0	0000					
170	05	CHEKNO	DCW	*		000	0	0					
180	06	CONTRL	DCW	*		000	0	0					
190	09	MASK1	DCW	*		,		0.					
200	06	MASK2	DCW	*		0	.						
201	07	MASK3	DCW	*		$	0	.					
202	22	MASK4	DCW	*		&	&						
203	11	MASK5	DCW	*			-	-					
204	01	ONE	DCW	*		1							
205			END	START									

Figure 56. Symbolic Program for Problem 8 (Part 5 of 5)

FARGO (Fourteen-O-One Automatic Report Generating Operation). This is a programming system for 1401 card systems that permits the user to use statements based on the logic of the IBM 407 to specify a desired report. FARGO requires no knowledge of 1401 programming or of computer techniques. A person with enough knowledge to do a report job on the IBM 407 can use the FARGO language with little, if any, further training.

No attempt is made here to go into details on these programs except to say that they can be adapted to specific cases by following a few simple rules regarding the punching of control cards.

File Maintenance

One of your duties in the XYZ Company is to update the three master files mentioned, on a weekly basis. This was easy with punched card files. You simply punched cards reflecting changes to the existing master files and updated them in one of three ways by:

1. Adding a complete new card record.
2. Removing a complete old card record.
3. Substituting a new card record for an old card record with the same man number.

With punched card records these operations can be, and usually are, performed manually when the number of changes to the master file at any one time is small. When the volume is large enough to warrant a machine operation, a collator performs the operations.

Updating a magnetic tape file is not so simple for the obvious reason that individual records cannot be physically inserted, removed, or substituted in a tape file, as with punched cards. Magnetic tape files are updated, not by making changes in the file itself, but by writing an entirely new tape even though only one record is to be added, removed, or changed.

Records written on tape are separated from each other by approximately ¾ inch of blank (unrecorded) tape, called an *inter-record gap* (IRG). To conserve tape space, and to speed up tape reading and tape writing, records are often grouped together, end to end, to form blocks of records. Thus, tape records (information between two inter-record gaps) may consist of single (logical) records, or two or more logical records in succession as shown in Figure 57. The number of logical records in each block is called the *blocking factor.* Single-record blocks have a blocking factor of 1.

Tape blocking has many advantages but it does create certain programming problems. When a tape block is read, all the logical records in the block are brought into core storage. The processing instructions must be arranged to pick up the records in succession from the storage area in which the block was read. This deblocking can be handled:

1. By moving the records from the block input area to a working storage area as needed, or
2. By using an index register (special feature) to select the records in succession from the block input area.

In writing, the records must be assembled (or blocked) in the output area, and then written when a complete block is assembled. Once again, this can be done either with a working-storage area or with an index register.

Problem 9. Updating a Tape File

We now present a problem involving the reading and writing of magnetic tape. In order to keep it simple, all records read and written are unblocked (have a blocking factor of 1).

This problem involves the updating of the employee name master tape file in which the 80-character unblocked records are arranged in social security number order. The card format to update the file is the same as Figure 18. In addition, a type-of-change code is punched in column 1 of the change cards as follows:

Change Code	*Type of Change*
1	New employee (addition to the file)
2	Employee no longer on the payroll (deletion from the file)
3	Employee who has moved from one department to another, or whose marital status or tax code has changed (substitution of new record for old)

In theory this operation is very similar to a merge-match operation with a collator, and requires a high-low-equal compare feature. A control number (in this case social security number) is compared. If the number on the card record (B-address) is *lower* than the number on the tape record (A-address), it indicates that a *new record* is to be added to the tape file.

If the number on the card record is *higher* than the number on tape record, it indicates that the tape record should remain in the *file unchanged.*

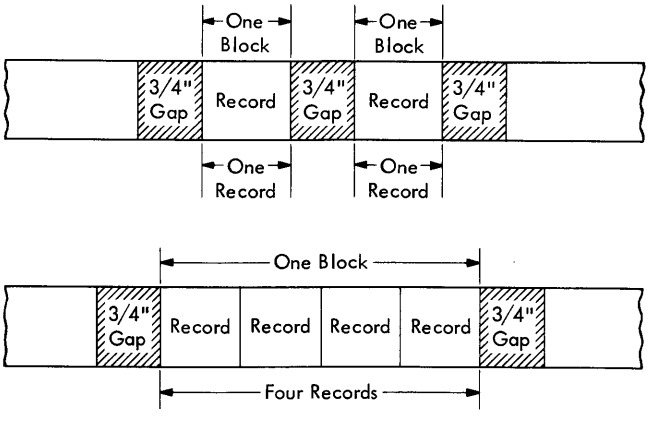


Figure 57. Single and Multiple Record Blocks

If the number on the card record is *equal* to the number on the tape record, one of two possible changes is indicated:

1. *Remove* the tape record from the file if the type-of-change code is two, or
2. Substitute the new card record for the old tape record if the type-of-change code is three.

Before developing a block diagram for this problem, refer to these sections in the 1401 reference manual.

Magnetic Tape

Tape Units—
- *IBM 729 Magnetic Tape Unit*
- *IBM 7330 Magnetic Tape Unit*
- *Tape Checking*
- *Magnetic Tape Operations*

Tape Instructions
- *Read Tape*
- *Write Tape*
- *Backspace Tape Record*
- *Write Tape Mark*
- *Skip and Blank Tape*
- *Rewind Tape*
- *Rewind Tape and Unload*
- *Branch if End of Reel*
- *Branch if Tape Error*

Block Diagram

Figure 58 shows a detailed block diagram of the computer operations, including card reading and tape handling. To understand best the actions this block diagram calls for, consider what you need to do when a *new* record is to be *added* to the file, what to do when an *old* record is to be *deleted* from the file, and what to do when an *old* record is to be *changed* for one reason or another or is to remain *unchanged.* How will we know whether a record is new or old, and if old, whether to be deleted, changed, or unchanged?

Obviously, no decision can be made until the first card record reads into the card read-in area (location 0001-0080), and the first tape record reads into an area of core storage decided on by the programmer. In this case, the area decided on according to the block diagram, is 0401-0480, since we are dealing here with 80-character records. It could just as well be 3901-3980, or any other area of 80-positions as long as the area chosen is not required by the stored program. A tape unit, on which a tape reel is mounted, can be manually set on any number from 0 to 9. If the tape unit is manually set at 1, the symbolic TAPE READ instruction in this case is:

Operation Code			
Mnemonic	*A-Address*	*B-Address*	*d-Character*
MU	%U1	0401	R

The mnemonic MU stands for *move unit.* The A-address %U1 refers to the tape unit set to 1. The B-address 0401 is the address into which the first character from each tape block is read. The d-character R (stands for *read*) specifies the way the tape unit is being used. Once a tape reel starts moving forward (or backward) the only thing that can stop it is an inter-record gap. Usually, all the characters within a tape block enter storage during the reading operation.

However, if a group-mark with a word-mark (BA8421M) is sensed in storage before the inter-record gap, the transmission of characters stops immediately. The remaining characters of the tape record continue to pass the reading unit until an inter-record gap is reached.

Similarly, when transmitting characters from storage to tape, you need a group-mark with a word-mark character in storage one position to the right of the last character to be transmitted to tape. Since, in this example, we are writing records sometimes from 0001-0080 and other times from 0401-0480, a group-mark with a word-mark is placed in 0081 and 0481 during the initialization phase of the program.

Once a card record and a tape record are in core storage, we can then determine whether the card record is lower or higher than, or equal to, the tape record by comparing the social security numbers. The number from the card is in location 0071-0079, and the number from the tape record is in location 0471-0479. Assume that the card is compared with the tape, using the instruction: C 0479 0079. The three conditions and actions to take are:

Social Security Numbers

Card (B)	*Tape (A)*	*Condition*	*Action*
077050990	077051123	Low B<A	*If code 1* in 0001, *write* card record on new tape *from 0001-0080,* and *read the next card.* If not code 1 HALT.
077051130	077051123	High B>A	*Write* old record on new tape *from 0401-0480* and *read next tape record.*
077051130	077051130	Equal B=A	*If code 2* in 0001, delete tape record by *reading* the *next card* and the next *tape record.* *If code 3* in 0001, *write* card record *from 0001-0080* and *read the next card and the next tape record.* If not code 2 or 3, HALT.

Because each of the three conditions requires a different course of action, we need three subroutines to make the necessary modifications in the tape-write routine. If the write-tape unit is manually set to 2 the symbolic tape WRITE instruction in this case is:

Operation Code			
Mnemonic	*A-Address*	*B-Address*	*d-Character*
MU	%U2	0x01	W

The main difference between this instruction and the READ TAPE instruction is the d-character W, which stands for WRITE. The A-address specifies tape unit 2. The B-address specifies the location of the high-order character of the record in storage. The hundreds position of the B-address (x) must be set to 0 or 4, depending upon whether writing is to occur from the card read-in area (0001), or the tape read-in area (0401).

Other modifications involve the setting of two program switches to control branching back to the main routine after writing on tape.

Tape-Error Routines

Tape operations usually include error-detection routines both in reading tape and writing tape. It's a standard feature of Input/Output Control Systems (IOCS) because the same problems face anyone who ever writes a tape program.

Tape errors are rarely the result of machine malfunction. Most are caused by the tape itself. The slightest imperfection in the tape can cause a character to read incorrectly. For this reason tape manufacturers subject tape reels to stringent inspection before shipping them to users.

Simple mechanical wear of the oxide coating on the tape, however, can cause the quality of the recorded signal to deteriorate. Moreover, dust particles can cause a weak signal to record, and great pains are taken by well-run installations to avoid dust and dirt on the tape. It is never wise, therefore, to assume that tape errors will not occur.

Figure 58 includes two tape-error routines. One is in connection with reading the old tape. The other is in connection with writing the new tape. The read-error routine provides for backspacing the tape record whenever an error is detected, and reading it as many as nine more times (if necessary) before the machine stops. If the tape error is the result of dust particles, backspacing and reading the same record several times often eliminates the trouble, and processing continues without interruption.

The write-error routine provides for the backspacing and writing of the same record in the same area of tape nine times. If the error still persists after nine attempts to write, a SKIP AND BLANK TAPE instruction (d-character E) erases about 3½ inches of tape. By erasing the bad area of tape, we get the effect of an unusually long inter-record gap. The idea is that the bad section of tape may be limited to one small area, and that tape farther along may be usable.

Program Halts

Figure 58 includes five program halts that call for some operator action before the operation can be resumed. It also includes a final halt which occurs when the operation is completed.

Halts 4 and 5 mean that the output reel has been rewound and unloaded, but the operation is not yet completed. Before operation can resume a new output reel must be mounted on tape unit 2. Pressing the start key resumes operation.

Halts 1, 2, and 3 mean trouble of one kind or another. Because these halts are closed loops, you cannot resume operation by pressing the start key. Some action is required to restart. Normally, operators would follow instructions contained in written procedures covering each condition. These instructions may involve storage printout of the error record, storage scan, error correction from the console, elimination or correction of erroneously punched cards, etc., and manual restarting from some specified point in the program.

Some conditions may even require that the job be started from scratch. The whole point of this type of halt is to force the operator to do something other than pressing the start key in order to proceed. Some programs print messages before coming to a halt.

Halt 1, for example, means that a master tape record still has one or more invalid characters (nonparity errors) after nine attempts to read it. Assume that you are the operator. What would you do in this case? Would it do any good to start from scratch? Knowing where the error record is in storage, would you print it on paper so that you may look at it, and possibly correct it from the console?

If the printed record appears to be correct, how would you determine which character in the record is invalid? Remember that the machine internally corrects the parity of the invalid character by either adding or removing the check bit. Thus, although the character is valid in storage, the printed character may not be the same character that appeared on tape.

An invalid character within a name can often be corrected on the spot just as you would correct any misspelled word. But how would you go about correcting an invalid digit within a social security number, for example? And, finally having located and corrected the invalid character, at what point in the program would you manually restart? A correction procedure, of course, would supply answers to all of these questions to guide the operator.

Halt 2 occurs when a card record that results in a low-compare condition (new record) does not have the change code (1) in column 1. But just because the condition is low, you can't assume that the change

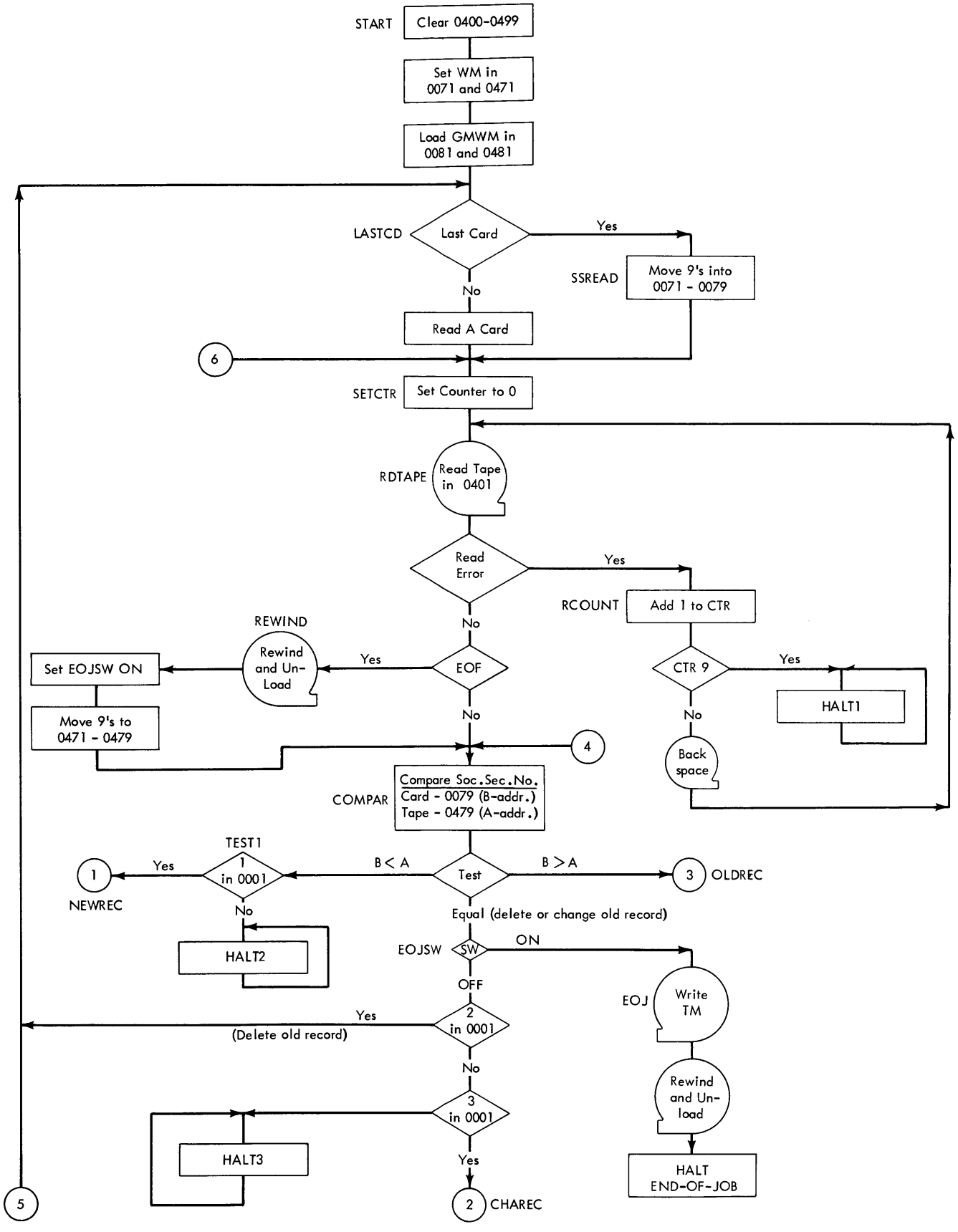


Figure 58. Block Diagram for Problem 9 (Part 1 of 2)

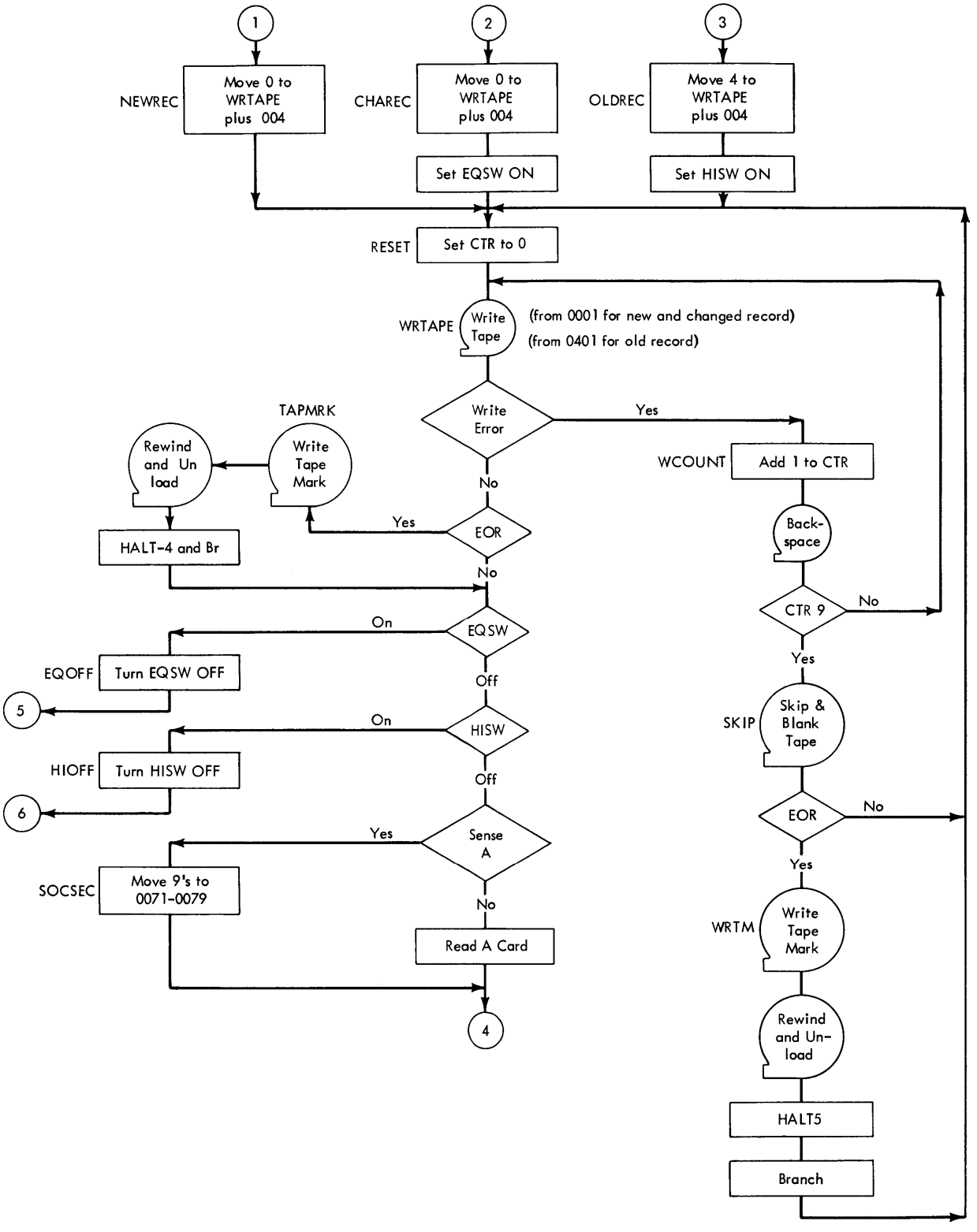


Figure 58. Block Diagram for Problem 9 (Part 2 of 2)

code is wrong. It could be that the change code is right, and the social security number is wrong. If investigation disclosed that the error is in the change code, the condition could be easily corrected on the console. The operation could be restarted from the proper point in the program without any difficulty.

But suppose it is found that the social security number is wrong, or that the card record is right but just simply out of sequence. Then what? You might better correct the condition one way or another, and start from scratch because if your change cards are out of sequence, it's a foregone conclusion that the tape records on the output tape will be out of sequence. A more sophisticated program would check the sequence of the records in the card file as well as the records in the input tape.

Labels used in the problem and their meanings appear in the following list in the order they appear on the block diagram. Labels followed by DCW* indicate constants with a word mark.

Label		*Meaning*
START		Start of program (origin 0500).
GMWM	DCW*	Group-mark with a word-mark character (card code 12-7-8).
LASTCD		Test for last card (start of main routine).
SSREAD		SOCIAL SECURITY READ (instruction to move a social security number of nine 9's into the read-in area, after reading the last card).
NINES	DCW*	Constant of nine 9's.
SETCTR		SET COUNTER (instruction to reset counter used for rereading and/or rewriting error-tape records).
ZERO	DCW*	Constant 0.
CTR	DCW*	Counter (counter for counting rereads and/or rewrites).
RDTAPE		READ TAPE (instruction to read tape record).
RCOUNT		READ COUNT (location of instruction to add 1 to counter).
ONE	DCW*	Constant 1 used to count.
REWIND		Location of the instruction to rewind the input tape.
EOJSW		End-of-job internal switch.
ON	DCW*	Constant B.
COMPAR		Location of the instruction to compare the social security number in 0071-0079 with the social security number in 0471-0479.
TEST1		Instruction to test whether a new record has a change code 1 in column 1.
EOJ		Start of the end-of-job routine.
NEWREC		Start of the new record routine.
CHAREC		Start of the change-record routine.
OLDREC		Start of the old record routine.
FOUR	DCW*	Constant 4.
EQSW		Equal switch (turned ON for change-record routine).
HISW		High switch (turned ON for old record routine).
RESET		Location of the instruction to reset the counter.
WRTAPE		WRITE TAPE (instruction to write tape record).
WCOUNT		WRITE COUNT (location of the instruction to add 1 to counter).
SKIP		Location of the instruction to SKIP AND BLANK TAPE (erase).
WRTM		Location of the instruction to WRITE TAPE MARK.
TAPMRK		Location of the instruction to WRITE TAPE MARK.
EQOFF		Location of the instruction to turn EQ-SW off.
OFF	DCW*	Constant N.
HIOFF		Location of the instruction to turn HI-SW off.
SOCSEC		SOCIAL SECURITY (location of the instruction to move a social security number of nine 9's into the read-in area after testing for last card (sense switch A on).

Figure 59 shows the symbolic program for *Problem 9.*

Autocoder With or Without IOCS

The 1401 *Autocoder* system is an advanced symbolic programming system that represents a significant extension of SPS-1 and SPS-2. The system is described in IBM *1401 Autocoder Specifications, J24-1434.*

The basic ideas of *Autocoder* are generally the same as for SPS. Like SPS, symbolic addresses may be used in place of numerical addresses. Mnemonic operation codes replace actual. The symbolic source program is translated into an actual object program by an assembly process.

The major differences between *Autocoder* and SPS are:

1. The *Autocoder* has a free-form coding sheet that has no fixed columns for operands A and B, and for the d-modifier. Instead the programmer uses as much space as required for each operand, and separates the operands by commas when there is more than one. High-order zeros in actual address

(0071, 0079, etc.) are omitted. Instruction comments may be written anywhere in the operand field of an *Autocoder* statement, if at least two spaces separate it from the last operand in the instruction.

2. Augmented mnemonic operation codes relieve the programmer of having to write the d-character in most instructions, and make it unnecessary to write the %U (for instance) in the A-address of a tape instruction.
3. It is possible to use *literals*. That is, instead of writing the *address* of a constant in an instruction, you may write the *constant* itself.
4. *Macro-instructions* are provided. That is, one instruction in the source program can be used to specify a whole routine consisting of many actual machine instructions.

The *Autocoder* system can be used with or without macro-instructions. In actual practice, liberal use is made of macro-instructions to reduce the number of detail instruction steps as much as possible. For example, in Figure 58 notice that much of the program is taken up with input/output routines and tape-error routines. Because these routines are essentially the same for most problems, writing them in detail for each program is a waste of time. Fortunately, a standard programming system is available called the IBM 1401 Input/Output Control System, or IOCS, that handles all of the normal input and output programming considerations and error routines with a minimum of programmer effort. The system is described in *IBM 1401 Input /Output Control System, C24-1462.*

Two other solutions to *Problem* 9 illustrate the use of the *Autocoder* system with and without macro-instructions.

Figure 60 is the *Autocoder* program without macro-instructions. The program steps are based on the block diagram in Figure 58 and are practically the same as for the SPS program in Figure 59. The same labels are used in both figures. The labels INPUT and OUTPUT used in *Autocoder* are *equated* to %U1 (input tape) and %U2 (output tape).

Figure 61 is the *Autocoder* program with IOCS. Of the many macro-instructions available in IOCS, the four basic macros are OPEN, GET, PUT, and CLOSE. They provide linkage to the IOCS library routines that read, write, block, deblock, and check records without further programming on the part of the user. Obviously, the use of macros simplifies the construction of the block diagram since the detail blocks normally shown for input/output and error routines are omitted.

Because program halts are an integral part of IOCS, halts 1, 4, and 5 in Figure 58 no longer apply. Several IOCS halts are generated instead. Each halt can be identified by displaying either the B-address register or the I-address register as indicated in Figure 32 of the 1401 IOCS manual. This figure lists the reason for each halt, and a procedure to follow if it occurs.

In addition, you have a choice of several alternate procedures if a parity error is detected when an input tape is read. The choice you select must be specified in a READERROR entry card as described in the IOCS manual.

Entries required in IOCS are of two main types:

DIOCS *Descriptive IOCS* that describes generally the machine configuration and the files used in the program.

DTF *Define The File* that describes, in detail, an individual file. A DTF entry must be included for each file processed by IOCS: card files (input and/or output), tape files (input and/or output), and printer (output).

DIOCS and DTF entries required for *Problem* 9 are shown in Figure 61 (Part 1).

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program UPDATE EMPLOYEE MASTER FILE

Page No. 01 of 4

Programmed by _____ Date _____

Identification PROB9

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND	d	COMMENTS
3 5	6 7	8 13	14 16	17 23			27	28 34			38	39	40 55
010			CTL	3,3									
020		* UPDATE	TF	EMPLOYEE MASTER FILE									
030			ORG	0,5,0,0									INITIALIZATION
040		START	CS	0,4,9,9									ROUTINE
050			SW	0,0,7,1				0,4,7,1					
060			LCA	GMWM				0,0,8,1					LOAD GROUP MARK-
070			LCA	GMWM				0,4,8,1					WORD MARKS
080		LASTCD	B	SSREAD								A	BR LAST CARD
090			R										READ A CARD
100		SETCTR	MCW	ZERO				CTR					SET CTR TO ZERO
110		RDTAPE	MU	%U1				0,4,0,1				R	STORE TAPE REC
120			B	RCOUNT								L	BR IF TAPE ERROR
130			B	REWIND								K	BR IF EOR
140		COMPARE	C	0,4,7,9				0,0,7,9					COMPARE SOC SEC#
150			B	TEST1								T	BR IF CARD LOW
160			B	OLDREC								U	BR IF CARD HIGH
170		EOJSW	NOP	EOJ									NOP-OFF B-ON
180			B	LASTCD				0,0,0,1				2	BR IF OLD RECORD
190			B	CHAREC				0,0,0,1				3	BR IF EQUAL
200		HALT3	H	0,0,0,0				0,0,0,3					STOP WRONG CODE
201			B	HALT3									

Figure 59. Symbolic Program for Problem 9 (Part 1 of 4)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program UPDATE EMPLOYEE MASTER FILE

Page No. 02 of 4

Programmed by _____ Date _____

Identification PROB9

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND	d	COMMENTS
3 5	6 7	8 13	14 16	17 23			27	28 34			38	39	40 55
010		CHAREC	MN	ZERO				WRTAPE	+	0,0,4			ADJUST B-ADDRESS
020			MCW	ON				EQSW					TURN SW ON
030		RESET	MCW	ZERO				CTR					RESET COUNTER
040		WRTAPE	MU	%U2				0,0,0,1				W	WRITE RECORD
050			B	WCOUNT								L	BR IF TAPE ERROR
060			B	TAPMRK								K	BR IF EOR
070		EQSW	NOP	EQOFF									NOP-OFF B-ON
080		HISW	NOP	HIOFF									NOP-OFF B-ON
090			B	SOCSEC								A	BR LAST CARD
100			R	COMPAR									
110		*											
120		* BRANCH	CH	SUBROUTINES									
130		*											
140		SSREAD	MCW	NINES				0,0,7,9					CHANGE SOCSEC NO
150			B	SETCTR									
160		RCOUNT	A	ONE				CTR					ADD 1 TO COUNTER
170			B	HALT1				CTR				9	BR IF CTR IS 9
180			CU	%U1								B	BACKSPACE ERROR
190			B	RDTAPE									REREAD TAPE REC
200		HALT1	H	0,0,0,0				0,0,0,1					STOP BAD RECORD
201			B	HALT1									

Figure 59. Symbolic Program for Problem 9 (Part 2 of 4)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program UPDATE EMPLOYEE MASTER FILE Page No. 03 of 4

Programmed by _____ Date _____ Identification PROBG

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND	d	COMMENTS
0010		REWIND	CU	%U1								U	REWIND-UNLOAD
0020			MCW	ON				EOJSW					TURN EOJSW ON
0030			MCW	NINES				0479					CHANGE SOCSEC NO
0040			B	COMPAR									BR TO COMPAR
0050		TEST1	B	NEWREC				0001				1	BR IF NEW RECORD
0060		HALT2	H	0000				0002					STOP-NEWREC NOT 1
0070			B	HALT2									
0080		NEWREC	MN	ZERO				WRTAPE	+	004			ADJUST B-ADDRESS
0090			B	RESET									BR TO WR ROUTINE
0100		OLDREC	MN	FOUR				WRTAPE	+	004			ADJUST B-ADDRESS
0110			MCW	ON				HISW					TURN HISW ON
0120			B	RESET									BR TO WR ROUTINE
0130		TAPMRK	CU	%U2								M	WRITE TAPE MARK
0140			CU	%U2								U	REWIND-UNLOAD
0150			H	0000				0004					STOP MOUNT REEL
0160			B	EQSW									
0170		EQOFF	MCW	OFF				EQSW					TURN EQSW OFF
0180			B	LASTCD									
0190		HIOFF	MCW	OFF				HISW					TURN HISW OFF
0200			B	SETCTR									

Figure 59. Symbolic Program for Problem 9 (Part 3 of 4)

IBM

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401 SYMBOLIC PROGRAMMING SYSTEM
CODING SHEET

FORM X24-1152
PRINTED IN U.S.A.

Program UPDATE EMPLOYEE MASTER FILE Page No. 04 of 4

Programmed by _____ Date _____ Identification PROBG

LINE	COUNT	LABEL	OPERATION	(A) OPERAND ADDRESS	±	CHAR. ADJ.	IND	(B) OPERAND ADDRESS	±	CHAR. ADJ.	IND	d	COMMENTS
0010		SOCSEC	MCW	NINES				0079					CHANGE SOCSEC NO
0020			B	COMPAR									BR TO COMPAR
0030		WCOUNT	A	ONE				CTR					ADD 1 TO COUNTER
0040			CU	%U2								B	BACKSPACE ERROR
0050			B	SKIP				CTR				9	BR IF COUNTER 9
0060			B	WRTAPE									
0070		SKIP	CU	%U2								E	SKIP & BLNK TAPE
0080			B	WRTM								K	BR IF EOR
0090			B	RESET									
0100		WRTM	CU	%U2								M	WRITE TAPE MARK
0110			CU	%U2								U	REWIND-UNLOAD
0120			H	0000				0005					STOP MOUNT REEL
0130			B	RESET									
0140		EOJ	CU	%U2								M	WRITE TAPE MARK
0150			CU	%U2								U	REWIND-UNLOAD
0160		FINSTP	H	FINSTP									JOB FINISHED
0170		* COMMENT		STANTS									
0180	01	GMWM	DCW	*		=							
0190	09	NINES	DCW	*		999999999							
0200	01	ZERO	DCW	*		0							
0201	01	CTR	DCW	*		0							
0202	01	ONE	DCW	*		1							
0203	01	ON	DCW	*		B							
0204	01	OFF	DCW	*		N							
0205	01	FOUR	DCW	*		4							
0206			END	START									

Figure 59. Symbolic Program for Problem 9 (Part 4 of 4)

IBM UPDATE EMPLOYEE

Program MASTER FILE

Programmed by _____

Date _____

INTERNATIONAL BUSINESS MACHINES CORPORATION

IBM 1401, 1410 AND 1440 DATA PROCESSING SYSTEMS

AUTOCODER CODING SHEET

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

Identification PROB 9 (76–80)

Page No. 001 (1 2) of 4

Line	Label	Operation	OPERAND
0.1		JOB	UPDATE EMPLOYEE MASTER FILE
0.2		CTL	33
0.3		ORG	500 INITIALIZATION ROUTINE
0.4	START	CS	499
0.5		SW	7.1,47.1
0.6		MLCWA	GMWM,8.1 LOAD WORD MARK
0.7		MLCWA	GMWM,48.1 LOAD WORD MARK
0.8	LASTCD	BLC	SSREAD BRANCH IF LAST CARD
0.9		R	READ A CARD
1.0	SETCTR	MLC	@0@,CTR SET COUNTER TO ZERO
1.1	RDTAPE	RT	INPUT,40.1 READ TAPE RECORD
1.2		BER	RCOUNT BRANCH IF TAPE ERROR
1.3		BEF	REWIND BRANCH IF END OF REEL
1.4	COMPAR	C	47.9,79 COMPARE SOCIAL SECURITY NUMBER
1.5		BL	TESTI BRANCH IF LOW COMPARE
1.6		BH	OLDREC BRANCH IF HIGH COMPARE
1.7	EOJSW	NOP	EOJ NOP-OFF B-ON
1.8		BCE	LAST,1,2 BRANCH IF OLD RECORD
1.9		BCE	CHAREC,1,3 BRANCH IF EQUAL
2.0	HALT3	H	0,3 STOP FOR WRONG CODE
2.1		B	HALT3
2.2	CHAREC	MN	@0@,WRTAPE+4 ADJUST B-ADDRESS
2.3		MLC	ON,EQSW TURN SWITCH ON
2.4	RESET	MLC	@0@,CTR RESET COUNTER
2.5	WRTAPE	WT	OUTPUT,1 WRITE RECORD FROM 1-80 OR 40.1-48.0

Figure 60. *Autocoder* for Problem 9 (Part 1 of 4)

IBM UPDATE EMPLOYEE

Program MASTER FILE

Programmed by _____

Date _____

INTERNATIONAL BUSINESS MACHINES CORPORATION

IBM 1401, 1410 AND 1440 DATA PROCESSING SYSTEMS

AUTOCODER CODING SHEET

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

Identification PROB 9 (76–80)

Page No. 002 (1 2) of 4

Line	Label	Operation	OPERAND
0.1		BER	WLCOUNT BRANCH IF TAPE ERROR
0.2		BEF	TAPMRK BRANCH IF END OF FILE
0.3	EQSW	NOP	EQOFF NOP-OFF B-ON
0.4	HISW	NOP	HIOFF NOP-OFF B-ON
0.5		BLC	SOCSEC BRANCH IF LAST CARD
0.6		R	COMPAR READ AND BRANCH TO COMPAR
0.7	*		
0.8	* BRANCH SUBROUTINES		
0.9	*		
1.0	SSREAD	MLC	NINES,79 CHANGE SOCIAL SECURITY NUMBER
1.1		B	SETCTR
1.2	RCOUNT	A	ONE,CTR ADD 1 TO COUNTER
1.3		BCE	HALTI,CTR,9 BRANCH IF COUNTER IS 9
1.4		BSP	INPUT BACKSPACE IF READ ERROR
1.5		B	RDTAPE REREAD TAPE RECORD
1.6	HALTI	H	0,1 STOP BAD RECORD
1.7		B	HALTI
1.8	REWIND	RWU	INPUT REWIND INPUT TAPE AND UNLOAD
1.9		MLC	ON,EOJSW TURN SWITCH ON
2.0		MLC	NINES,47.9 CHANGE SOCIAL SECURITY NUMBER
2.1		B	COMPAR BRANCH TO COMPAR
2.2	TESTI	BCE	NEWREC,1,1 BRANCH IF NEW RECORD
2.3	HALT2	H	0,2 STOP NEW RECORD NOT 1
2.4		B	HALT2
2.5	NEWREC	MN	@0@,WRTAPE+4 ADJUST B-ADDRESS

Figure 60. *Autocoder* for Problem 9 (Part 2 of 4)

IBM UPDATE EMPLOYEE

Program MASTER FILE

Programmed by _____

Date _____

INTERNATIONAL BUSINESS MACHINES CORPORATION

IBM 1401, 1410 AND 1440 DATA PROCESSING SYSTEMS

AUTOCODER CODING SHEET

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

Identification PROB9

Page No. 03 of 4

Line	Label	Operation	OPERAND
0.1		B	RESET BRANCH TO WRITE ROUTINE
0.2	OLDREC	MN	FOUR,WRTAPE+4 ADJUST B-ADDRESS
0.3		MLC	ON,H1SW TURN SWITCH ON
0.4		B	RESET BRANCH TO WRITE ROUTINE
0.5	TAPMRK	WTM	OUTPUT WRITE TAPE MARK
0.6		RWU	OUTPUT REWIND OUTPUT TAPE AND UNLOAD
0.7		H	0,4 STOP MOUNT REEL
0.8		B	EQSW
0.9	EQOFF	MLC	OFF,EQSW TURN SWITCH OFF
1.0		B	LASTCD
1.1	H1OFF	MLC	OFF,H1SW TURN SWITCH OFF
1.2		B	SETCTR
1.3	SOCSEC	MLC	MINES,99 CHANGE SOCIAL SECURITY NUMBER
1.4		B	COMPAR BRANCH TO COMPAR
1.5	WCOUNT	A	ONE,CTR ADD 1 TO COUNTER
1.6		BSP	OUTPUT BACKSPACE ERROR
1.7		BCE	SKIP,CTR,9 BRANCH IF COUNTER IS 9
1.8		B	WRTAPE
1.9	SKIP	SKP	OUTPUT SKIP AND BLANK TAPE
2.0		BEF	WRTM BRANCH IF END OF REEL
2.1		B	RESET
2.2	WRTM	WTM	OUTPUT WRITE TAPE MARK
2.3		RWU	OUTPUT REWIND OUTPUT TAPE AND UNLOAD
2.4		H	0,5 STOP MOUNT REEL
2.5		B	RESET

Figure 60. *Autocoder* for Problem 9 (Part 3 of 4)

IBM UPDATE EMPLOYEE

Program MASTER FILE

Programmed by _____

Date _____

INTERNATIONAL BUSINESS MACHINES CORPORATION

IBM 1401, 1410 AND 1440 DATA PROCESSING SYSTEMS

AUTOCODER CODING SHEET

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

Identification PROB9

Page No. 04 of 4

Line	Label	Operation	OPERAND
0.1	EOT	WTM	OUTPUT WRITE TAPE MARK
0.2		RWU	OUTPUT REWIND OUTPUT TAPE AND UNLOAD
0.3	FINSTP	H	FINSTP JOB FINISHED
0.4	*		
0.5	* CONSTANTS		
0.6	*		
0.7	GMWM	DCW	@$@
0.8	NINES	DCW	@999999999@
0.9	CTR	DCW	@0@
1.0	ONE	DCW	@1@
1.1	ON	DCW	@8@
1.2	OFF	DCW	@N@
1.3	FOUR	DCW	@4@
1.4	INPUT	EQU	%U1
1.5	OUTPUT	EQU	%U2
1.6		END	START

Figure 60. *Autocoder* for Problem 9 (Part 4 of 4)

IBM UPDATE EMPLOYEE
Program MASTER FILE
Programmed by
Date
INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401, 1410 AND 1440 DATA PROCESSING SYSTEMS
AUTOCODER CODING SHEET
Form X24-1350-
Printed in U.S.A.
Identification PROBG (76-80)
Page No. 01 (1-2) of 3

Line (3-5)	Label (6-15)	Operation (16-20)	OPERAND (21-70)
0.1		JOB	UPDATE EMPLOYEE MASTER FILE
0.2		CTL	33
0.3		DIOCS	
0.4	IODEVICES		READER,TAPE
0.5	READERROR		SCAN
0.6	RWDOPTION		UNLOAD
0.7	DIOCSORG		SDO
0.8		DTF	CARD
0.9	CARDPOC		1
1.0	EOFADDR		SSREAD
1.1	FILETYPE		READER
1.2		DTF	INPUT
1.3	CHANDRIVE		1
1.4	EOFADDR		REWIND
1.5	FILETYPE		TAPE,INPUT
1.6	IOAREAS		TAPIN
1.7	REWIND		UNLOAD
1.8	SIZEREC		80
1.9		DTF	OUTPUT
2.0	CHANDRIVE		2
2.1	FILETYPE		TAPE,OUTPUT
2.2	IOAREAS		TAPOUT
2.3	REWIND		UNLOAD
2.4	SIZEREC		80
2.5			

Figure 61. *Autocoder* and IOCS for Problem 9 (Part 1 of 3)

IBM UPDATE EMPLOYEE
Program MASTER FILE
Programmed by
Date
INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401, 1410 AND 1440 DATA PROCESSING SYSTEMS
AUTOCODER CODING SHEET
Form X24-1350-
Printed in U.S.A.
Identification PROBG (76-80)
Page No. 02 (1-2) of 3

Line (3-5)	Label (6-15)	Operation (16-20)	OPERAND (21-70)
0.1	START	SW	7,1 SET WORD MARK
0.2		OPEN	CARD,INPUT,OUTPUT OPEN ALL FILES
0.3	RDCD	GET	CARD READ A CARD
0.4	RDTAPE	GET	INPUT READ A TAPE RECORD
0.5	LOOFF	MCW	@N@,LOSW TURN LOW SWITCH OFF
0.6		C	SSEC,79 COMPARE SOC. SEC. NO. CARD-TO-TAPE
0.7		BL	TEST1 BRANCH I.F. LOW COMPARE
0.8		BH	OLDREC BRANCH I.F. HIGH COMPARE
0.9	EOJSW	NOP	EOJ EQUAL COMPARE NOP-OFF B-ON
1.0		BCE	RDCD,1,2 DELETE CODE-2 RECORDS
1.1		BCE	CHAREC,1,3 PROCESS CODE-3 RECORDS
1.2	HALT3	H	0,3 STOP I.F. EQUAL CARD NOT CODE-2 OR
1.3		B	HALT3 CODE-3
1.4	*		
1.5	* BRANCH SUBROUTINES		
1.6	*		
1.7	TEST1	BCE	NEWREC,1,1 BRANCH I.F. LOW CARD IS CODE-1
1.8	HALT2	H	0,2 STOP I.F. LOW CARD IS NOT CODE-1
1.9		B	HALT2
2.0	NEWREC	PUT	CARDIN,OUTPUT LOW CARD CODE-1, WRITE NEW RECORD
2.1		MCW	@B@,LOSW TURN LOW SWITCH ON
2.2		GET	CARD READ A CARD
2.3		B	LOOFF
2.4	OLDREC	PUT	TAPIN,OUTPUT CARD HIGH - TAPE LOW - OLDRECORD
2.5		B	RDTAPE

Figure 61. *Autocoder* and IOCS for Problem 9 (Part 2 of 3)

IBM UPDATE EMPLOYEE
Program MASTER FILE
Programmed by _____
Date _____

INTERNATIONAL BUSINESS MACHINES CORPORATION
IBM 1401, 1410 AND 1440 DATA PROCESSING SYSTEMS
AUTOCODER CODING SHEET

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

Identification PROB9 (76 80)
Page No. 03 (1 2) of 3

Line (3 5)	Label (6 15)	Operation (16 20)	OPERAND (21 25 30 35 40 45 50 55 60 65 70)
0.1.	CHAREC	PUT	CARDIN,OUTPUT
0.2.		B	RDCD
0.3.	EOJ	CLOSE	OUTPUT
0.4.	FINSTP	H	FINSTP
0.5.	*		
0.6.	*EOFROUT	INES	FROM IOCS
0.7.	SSREAD	CLOSE	CARD
0.8.		MCW	NINES,79
0.9.	LOSW	NUP	LOOFF
1.0.		B	RDTAPE
1.1.	REWIND	CLOSE	INPUT
1.2.		MCW	@@@,EOJSW
1.3.		MCW	NINES,SSEC
1.4.		B	LOOFF
1.5.	*		
1.6.	*CONS	TAN	TS
1.7.	NINES	DCW	@9999999999@
1.8.	TAPOUT	DA	1X80,G,C
1.9.		ORG	1
2.0.	CARDIN	DA	1X80,G
2.1.		ORG	401
2.2.	TAPIN	DA	1X80,G,C
2.3.	SSEC		71,2,79
2.4.		END	START
2.5.			

Figure 61. *Autocoder* and IOCS for Problem 9 (Part 3 of 3)

Index

IBM 1401 Printed in U.S.A. C24-1494-2

IBM

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