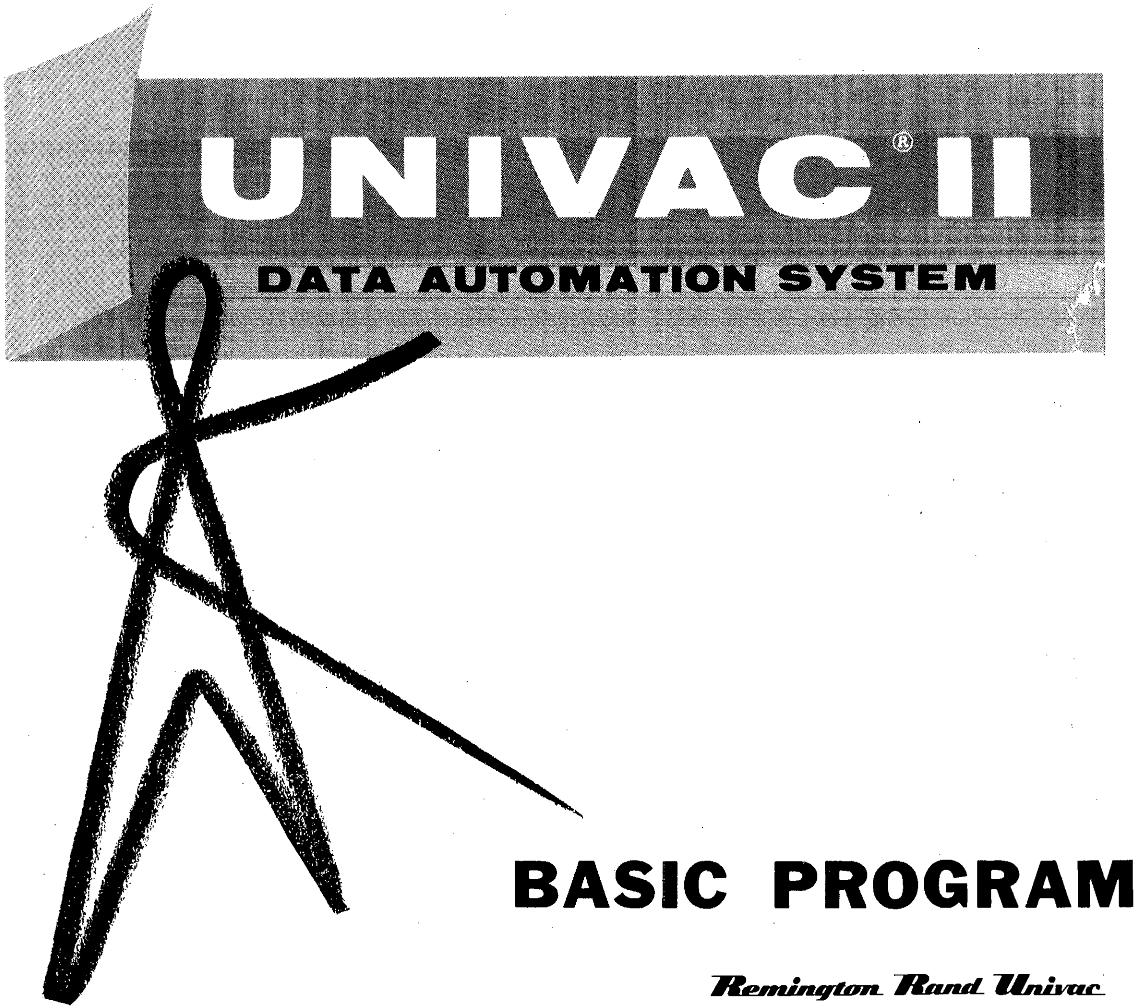


UNIVAC® II

DATA AUTOMATION SYSTEM

BASIC PROGRAMMING

Remington Rand Univac
DIVISION OF SPERRY RAND CORPORATION

Preface

This manual is a basic text in programming the Univac II Data-Automation System. It is an introduction to the foremost data processing system yet developed, and to dynamic and advanced concepts in the field of data processing.

Study of this manual will also acquaint you with the language of the Univac II Data Automation System, and the principles of efficient Systems Design so that maximum utilization of the capabilities of the Univac II Data Automation System can be achieved.

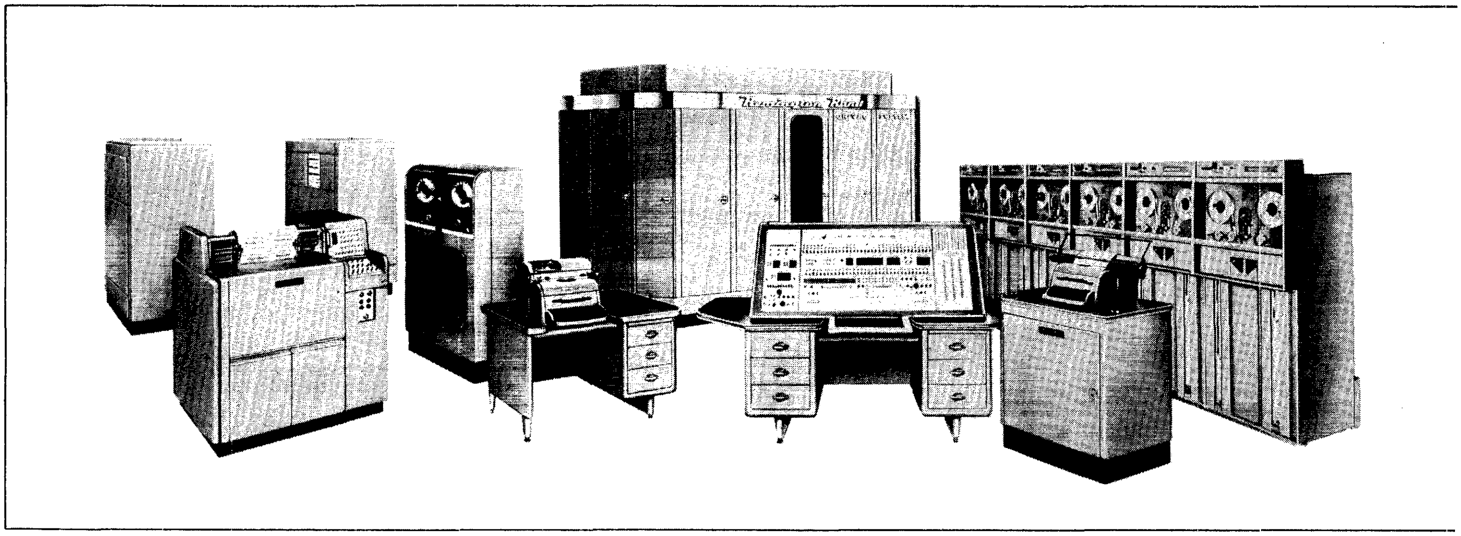
Although intended primarily as a text for a course of instruction conducted by the Remington Rand Univac Training Department, it is recognized that many may not have the opportunity to attend such a formal training course. With this in mind, each topic has been introduced with a thorough explanation followed by illustrative examples and student exercises.

This manual is for the beginner. No prior knowledge of programming, electronics, or data processing is necessary to its understanding. The prime requisite is a willingness to learn coupled with diligent application.

Table Of Contents

PAGE

PAGE


Remington Rand

chapter 1

Introduction

DATA PROCESSING AREAS IN BUSINESS

A first step in a study of electronic computers is to survey the areas of business operations wherein a computer may become a useful managerial tool. These areas are called data processing areas. In its day-to-day functioning a manufacturing concern is composed of myriad channels through which money and material flow in fulfillment of the company's obligations to its stockholders, employees, vendors, customer and the government. From a data processing point of view, these areas are concerned with management's attempts to record, measure and effectively control this flow. Because of its broad yet familiar activities the manufacturing company's activities will be considered. Figure 1-1 is a generalized block diagram of a typical manufacturing company and its environment.

The most common data processing areas have been indicated on the chart. A very brief description of each is listed below.

A TYPICAL MANUFACTURING ORGANIZATION AND ITS DATA PROCESSING ENVIRONMENT

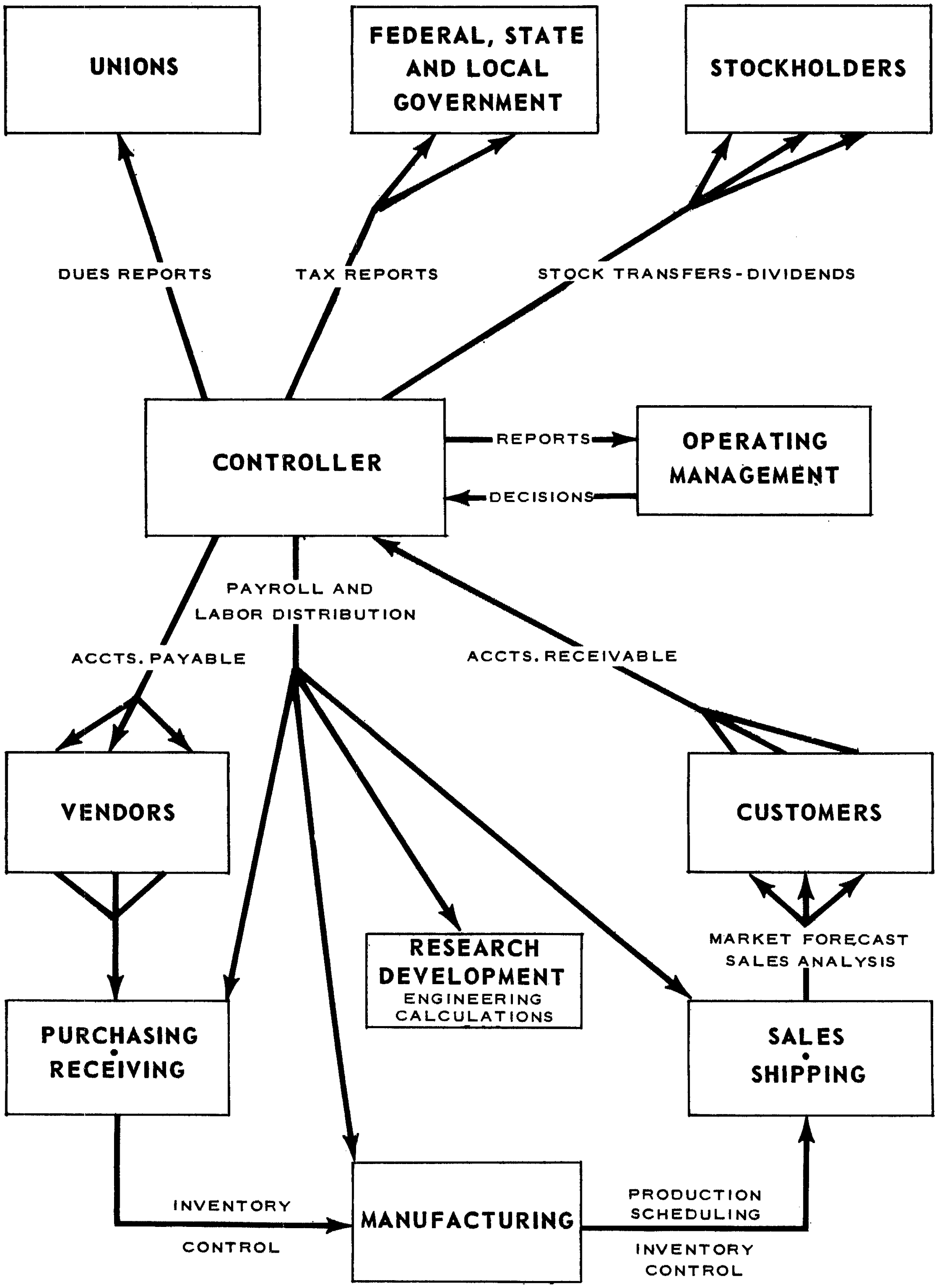


FIGURE 1-1

MARKET FORECASTING AND SALES ANALYSIS:

To attempt to find the beginning of the movement through the channels pictured would be to search for the beginning of a circle because of the multitudinous cross-references and interdependencies which exist. From the point at which planning for the next year commences, however, a certain sequence does follow.

At that point the big question is, "How good will the new year be?". The answer can be found by making a reliable sales forecast to serve as a basis upon which all operational planning will be laid.

The past sales history is essential to such a forecast. Thus, many concerns break down their sales as often as once per week according to the products sold, the regions in which they were sold, the dollar values of the sale, the percentage gross and/or net profit obtained and other significant criteria. In addition to predicating any immediate action which needs to be taken, such reports, if compiled over a period of years, will yield information on the seasonal and regional fluctuations of the sales of various products.

A further study of such a sales analysis may bring out some revealing correlation between the concern's sales and the general business trends and cycles, customer activities and similarly relevant factors. Such correlations are not always easy to find; but once discovered, they offer the means of making a reliable forecast of the sale of each product in each marketing area. An evaluation of the market forecast will affect the budget and production levels to be maintained during the year.

PRODUCTION SCHEDULING:

The sales forecast and any adjustments to it which may be necessary as the year progresses are the sources of the production orders. The production orders indicate the date of completion and size of each batch of every product to be manufactured. Referencing these orders against a bill of materials listing is then the basis of the production scheduling operation. This listing contains the material, machines and time required for the completion of each phase in the manufacture of the product. Working backwards from the "due date" it is possible to list the times at which materials and machines must be available if the due date is to be met. Proper planning is essential since any misscheduling of machine requirements may result in delays and extra production expense for overtime, or idle machinery and idle men. In addition to yielding a machine schedule, the bill of materials listing yields the requisitions for the total raw material requirements and the time in the production line at which they must be available.

INVENTORY CONTROL:

From the bill of materials listing, information is also obtained for the inventory control. As a by-product of the machine scheduling, the quantities of raw materials needed during each manufacturing phase are also determined. These raw material requirements are used for the publication of requisitions. In addition, they are compared to the current inventory level of the material and posted to it. If the reorder level is reached, production or purchasing orders, depending on whether the material is processed within the company or purchased, are issued in order to replenish the stock. Proper use of reorder levels can offer considerable savings by accurate control of the minimum inventory level to be maintained. Accurate inventory control is essential in reducing the capital investments and storage obsolescence costs of large inventories, or the costs of emergency reorders and delays resulting from shortages.

ACCOUNTS PAYABLE:

The accounts payable operation is initiated by the receipt of an invoice from the vendor. This invoice is first checked for amounts billed against quantities received and priced against the current price list. Then, although immediate payment of all accurate invoices is possible, payment is usually postponed temporarily to allow further use to be made of the available cash. Such unpaid invoices are listed on the accounts payable ledger. Cash balances and the efficient use of any discount privileges determine the time for selection from this ledger for payment. Checks are produced and appropriate entries made in the vendor's account. Information available may also be extracted for such things as general ledger and property accounting, and reports on vendor activity.

PAYROLL AND LABOR DISTRIBUTIONS:

This area is commonly the most highly mechanized data processing area in business today. In spite of the fact that all payrolls are designed primarily to produce paychecks, the variety in important payroll details caused by unusual or individual labor contracts, differing local and state regulations and plant policies preclude a complete uniformity of description. With this precautionary statement in mind, consider payroll data processing to be divided into three parts: determination of gross pay, computation of net pay from the gross, and labor distribution.

Determination of gross pay may be a trivial operation in the case of a salaried payroll. Here gross pay is a fixed, constant amount for each pay period agreed to in advance by the employee and employer. Usually no calculations are necessary, and the determination of gross pay amounts to simply calling for it from the employee's record. In most cases, however, the determination of gross pay is an involved process. Gross pay is more often based upon the number of hours worked in each of several hourly rate categories (regular and overtime factors) during the pay period. This may be modified in many plants to include bonus or efficiency payments determined by the output of groups of workers or by a piecework schedule. The net pay calculation involves the computation of tax deductions imposed by state, local and federal governments and such other deductions, usually variable in amount from one pay period to the next, as specified by union contracts and fringe benefits or employee options. The end product of the net pay calculations is a series of paychecks (or pay slips if payment is by cash) and various payroll registers listing gross and net pay and the several deductions. In addition to these, the individual earnings record must be updated for end-of-quarter and end-of-year government tax reports.

The labor distribution phase is used by management to establish product costs and selling prices. Gross-pay and hours-worked data for each employee, established in the gross pay phase, are distributed to each product, account or activity he has engaged in during the pay period. These are then summarized to produce labor costs for each of the distributed categories.

TAX REPORTS - UNION DUES REPORTS, ETC.:

Under present labor-management practices, management assumes many of the employee's obligations to his environment. Taxes, union dues and various voluntary deductions are withheld. The necessity arises for the firm to make reports to the government, union dues reports to the unions, hospitalization and insurance reports, etc. The information for such reports is available from the payroll processing itself.

Year-to-date totals of gross pay, income tax withheld and FICA tax are sufficient for the preparation of W-2 forms. Similarly a compilation from the employee files and payroll processing results is all that is necessary in the preparation of most other reports.

ACCOUNTS RECEIVABLE:

The accounts receivable operation commences when a shipping document is received indicating that delivery of an order has been made. Products listed on this document are priced and the shipment is extended to produce the invoice sent to the customer. At the same time, the total dollar charge is posted to the customer's records on the accounts receivable ledger.

This ledger is often scanned daily. Cash receipts and any earned discounts are credited to it. Appropriate information is entered into the customers' credit history. Ageing accounts are extracted, checked, their credit history examined and appropriate action is taken. At the end of the month, the information present is compiled to form monthly statements, which also may be sent to the customer.

STOCK DIVIDENDS AND TRANSFERS:

In addition to the data processing involved in the manufacturing company's obligations to supplier and customer, employees and government, some arise from its basic obligations to the stockholders, namely, allowing them a voice in the management and a share in the profits.

Up-to-date listings of holders are essential to the proper satisfaction of these obligations. Consequently the holder listings must be periodically maintained to assure that they reflect the latest results of all stock issues, cancellations and transfers. When a dividend is declared, it is then only necessary to select the owners as of that date from the listing and multiply the dividend rate by the number of their shares to make the proper disbursement. Similarly a scanning of this list is sufficient when it is necessary to print and distribute the proxy ballots for the annual stockholders meeting. Year-to-date dividends paid and other information on this listing may be employed in the preparation of the year-end state and federal tax reports and of any statistical reports desired.

PRESENT ELECTRONIC COMPUTER APPLICATIONS

In the preceding section a manufacturing company's operations were discussed to bring into focus some of the data processing areas in which electronic computers are being considered as a more effective managerial tool. In this section specific applications for which organizations have profitably employed computers are listed and classified by the type of industries.

This list is intended to be indicative and not exhaustive due to the constantly increasing number of installations and to the continuing expansion of applications at current installations.

ADVERTISING:

Calculation of sales volume predictions based upon past sales, employing least squares fit technique; monthly tabulation of editorial lineage by predetermined classifications.

AIRCRAFT MANUFACTURING:

Payroll, labor distribution, efficiency ratings; engineering and scientific problems including trajectory and matrix calculations,

CHEMICAL MANUFACTURING:

Payroll, engineering problems, central billing, inventory and production control, accounts receivable, sales statistics, solution of differential equations.

ELECTRICAL AND ELECTRONIC MANUFACTURING:

Production control including scheduling, inventory control, preparation of shipping documents, production scheduling and market forecast; financial control including payroll, preparation of invoices, cost and general ledger accounting and budget preparation.

GOVERNMENT AGENCIES:

HEADQUARTERS UNITED STATES AIR FORCE - Aircraft scheduling, budgeting, determination of supply requirements, calculations of projected aircraft engine inventory for several engine types.

AIR MATERIEL COMMAND - Payroll, procurement, supply, control -- fiscal and budget.

ATOMIC ENERGY COMMISSION - Problems arising in the design and construction of power and research reactors; problems in nuclear physics.

BUREAU OF CENSUS - Population statistics; seasonal business trends.

DEFENSE DEPARTMENT - Tables of phased requirements in personnel and material for the support of operational programs.

WEATHER BUREAU - The circulation acceleration of atmosphere at different levels computed from radiosonde data.

GENERAL INDUSTRIAL:

Payroll, cost distribution, stores accounting, market forecasting; production control including material scheduling, inventory control and production scheduling; general accounting including invoicing, cost and general ledger accounting, and budget preparation; capital accounting; stock transfer; prediction of the variability of product mixture based on changing components employing regression techniques; material utilization analysis; explosion and summarization of parts and sub-assemblies for production orders.

INSURANCE COMPANIES:

Premium billing and accounting, agents' commission payments, dividend calculation, policy loan accounting, actuarial computation, market analysis, annual statement preparation, mortgage loan billing, file maintenance, agency statistics, field auditing, policy loans, reserve valuation, payroll, check issue, actuarial statistics.

OIL COMPANIES:

Oil payment accounting, gas payment accounting, payroll, bulk station checking, sales and stock statistics, matrix calculations, calculations for optimization of amount of end product from raw input.

PUBLIC UTILITIES:

COMMUNICATIONS - Computation of minimum cost for the transmission of information through a nationwide network; summarization of data by over 700 classifications; computations of trend correlations.

LIGHT AND GAS - Billing, payroll, matrix calculations, computation of utility rate tables.

TRANSPORTATION -Revenue, material, vehicle accounting; payroll, personnel records, cost distribution, plant accounting.

PUBLISHING:

Advertising space and sales analysis; preparation of mailing list using statistical sampling technique.

RAILROADS:

Freight revenue accounting, payroll, capital accounting, stock transfer, labor distribution, I.C.C. reports.

DATA PROCESSING EQUIPMENT

In the preceding section, the general data processing areas in industry were briefly described, and some of the applications for which computers are currently being utilized were listed. To the beginning student of electronic data processing many of these areas must have appeared so unrelated to each other as to warrant objections to their being lumped together as being only different aspects of a common operation. Prior to the advent of electronic computers, management usually believed there were no common grounds between them -- that payroll and premium billing are too different to allow their study by general techniques. The desire to apply electronic computers to business problems required computer designers to search out the fact that payroll and premium billing were indeed susceptible to study by a universal technique. This means that a generalized description of data processing can be developed without the necessity of studying each individual area of application. Figure 1-2 is a general block diagram of a data processing system.

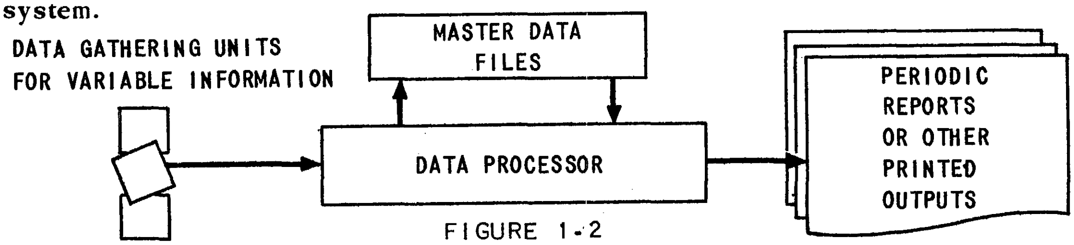


FIGURE 1-2

Each of the data processing areas described above consists of the four elements shown in Figure 1-2. A data processing system is best described by its outputs. These are the various reports, summaries, statistics, bills, checks, invoices, etc., either required by management or government, or which are a necessary part of the detailed operation of the company. The information required on these printed outputs and their frequency establish the general requirements for the remaining three elements.

The inputs to the data processing system from which the output reports are to be compiled usually consist of two types; master data that remains essentially unchanged from one reporting cycle to the next or which changes in a known and fixed way; and variable information which is produced by the unpredictable activities of the business. Master data is the permanent information records containing identifying and historical data about the individual, account, item, product or service being reported. Examples of master data are: names and addresses, account numbers, current credits and debits, running inventories, wear-out rates, etc. Variable information is data introduced to the data processing system reflecting current operations. It is generated by human activity and is thus essentially unpredictable. Examples of variable information are: the hours worked by an employee, receipts, expenditures, sales, shipments, etc. Since the transaction producing these variables are often physically dispersed (coming from different departments or branch offices) some means of gathering the data for injection into the system is required.

The data processor is the converter of master and variable data into the output reports. As the block diagram indicates it also posts changes, when necessary, to the master data files. These changes are introduced to the processor through essentially the same data gathering units which are used for the variable information.

The data processor may be a clerical staff or an electronic computer with all gradations between these two extremes. In order to be able to produce the output reports, the data processor must be able to

1. read documents,
2. record documents and reports,
3. sort and classify data,
4. calculate,
5. make simple decisions.

The comparison of traditional data processing systems and electronic computer systems which follows is intended to point out the similarities and differences in overall approach required by each type of equipment.

MANUAL METHODS:

The postwar growth in American business has been characterized by streamlining factory and production methods in order to achieve high production at a lower unit cost. Office management, however, has increased the data processing output required by this higher factory production more by simply increasing the clerical force than by improving unit costs through cleaning up or redesigning the data processing system. Admittedly system redesign is a difficult and involved task, requiring a careful study of each data processing system extant so that overlapping or useless operations may be removed.

For low volume work the human being has the considerable advantage of requiring no translation of business records or transactions, and his methods can be changed with relative ease. He is, however, susceptible to error especially if the work is highly routine or if the work load is heavy; his speed of calculating, posting and printing is slow; and for large volume work his cost is disproportionately high.

KEY DRIVEN DEVICES:

The next step in the direction of the automatic office is in improving a manual system by adding key-driven devices. This includes such standard office equipment as typewriters, adding machines, calculators and accounting machines. In general, these devices mechanize the calculation and printing functions of the data processor. In their simplest form, these key driven devices are either a typewriter performing the printing function only or an adding machine performing addition or subtraction functions. By appropriately combining the print and calculate functions, and further, by building in a more elaborate control mechanism which can govern addition and subtraction, a class of calculators capable of performing all four arithmetic operations and the automatic printing of results has been produced and is in common use.

A further extension of the printing calculator principle to provide greater flexibility in the format of printed results has led to the typical accounting machine. A trend of very recent origin is the connection to these accounting machines of paper tape punches which provide a means of direct communication to other types of calculating and computing mechanisms without the need for re-transcription of data. These devices are members of a common language data processing system.

For moderate volumes of data involving relatively short sequences of calculations, unit cost is low, speed and accuracy is greater than manual methods, and trained and experienced personnel are available. Changes in procedure and forms can be accommodated with relative ease. Machines with punched paper tape output provide a means for further calculations to be done without the necessity for manual re-entry of data.

One disadvantage of key driven devices is that data translation to a machine readable form is necessary, with translation consisting of the entry of data into a keyboard. Another disadvantage is that the sorting and classifying of data, decision making and the sequencing of operations must be done by human intervention.

PUNCHED CARD MACHINES:

Prior to 1953 punched card equipment, commonly referred to as tabulating equipment, marked the apex of office automation. The general plan of tabulating equipment is to provide specialized machines which perform each of the five basic elements of the requirements listed for a data processor, although in some cases a single machine may perform several such functions. These machines are sorters, tabulators, calculators, interpreters, collators, key punches and reproducers. These machines form a common language group. The punched card serves as a data communication and storage medium.

As compared with key-punch or manual methods punched card machines have speed, accuracy, and low unit cost for relatively large volumes of work as well as permitting variety and completeness in reports. Changes in procedure and forms may be relatively difficult, and the scope of operations performed is limited so that exceptions must be handled manually. There is in addition the arbitrary limitation of record size to either 80 or 90 digits or multiples of 80 or 90 digits.

ELECTRONIC COMPUTER CLASSIFICATIONS:

IN-LINE COMPUTERS:

Although the logical roots of modern electronic computers extend back over a century, the advent of their application to office automation was in 1953. Since then, the development of electronic data processors (or computers for short) has proceeded in two main directions: the "in-line" or "real-time" computers contrasted with "off-line" or "delayed" computers. The distinction is primarily a matter of application rather than the computing equipment itself, since the determinant of the mode in which any particular computer will be employed is strictly one of economics. Considerations of economic design has led to the development of computers intended for but not restricted to each mode, and this has often led to the equipment itself being identified as in-line or off-line.

An in-line application is one in which transaction information reflecting current activity is introduced into the data processor as it occurs resulting in the immediate modification of the master data record. Thus, these master records are always up to date. Thus modification of these master records is in effect directly ''in the line'' of the company's operations.

The basis for the in-line computer is the development of relatively high-speed storage facilities of exceptionally large capacities capable of storing the entire master file of data processing area. Further, this large capacity file storage must make available to the other components of the computer its stored information at a very rapid rate.

Once the existence of this type of storage became available, it was possible to connect the data gathering units to the computer so that transactions could be entered directly into the computer, which can then consult the master file informa-with minimum delay. The calculations or other processing is done on an item-by-item basis, that is, on demand. Through the use of a stored program principle, the computer can sort and classify data, perform long sequences of involved calculations, and can exercise simple decisions in a completely automatic manner. Auxiliary units attached to the computer can be used to print final results or reports of such calculations if desired.

An example of this type of computer is the airline reservations system. The Univac File Computer is an example of equipment which can be applied to a variety of in-line operations. An obvious advantage of in-line computers is that it permits the coupling of ''point of sale recording'' with direct inquiry units. Where fully current information in readily available form on each account or item is essential, the in-line computer offers high-speed and accuracy of operation. Another feature is that the task of pre-sorting the batches of transaction data (which is a characteristic of the off-line applications) is unnecessary with these so-called random access memories.

One limitation of these applications is that the volume of transactions of a particular type must justify a special piece of equipment. However, most business applications will require that the electronic system perform other operations than simply the modification of master records by the posting of transactions - such operations as records of the posting, order issuance and summary report preparation. Hence, this equipment must be made adaptable for off-line operations as well as in-line.

Another problem which may arise with these computers is caused by the expansion and contraction of the master file. If an electronic device is purchased of just the size necessary to store the current master file, difficulty arises when adding new items to the file, especially when a record must be fitted to an already full section. Contrasted to this, the deletion of items causes gaps. If a device is purchased sufficiently large to hold all possible numbers, a great many more critical gaps may occur in the areas of inactive and non-existent items.

In general the in-line computer is inflexible in the numbering or identifying of data for the items in the master file, and a further difficulty is that there are no provisions for the searching of files by other than the key criterion on which the items have been stored. For example, the master records in a stock inventory application may be stored by the stock number key, yet an inquiry or a summarization may be required on a selected supplier's code.

The major subject of this manual is the application of the off-line computer, although many of the techniques illustrated are applicable to in-line computers.

OFF-LINE COMPUTERS

A major area of data automation interest is in the application of the off-line electronic computer. Instead of handling each transaction as it occurs (as would be the case for the in-line computer) transactions are batched until an economical work load is gathered. These batched transactions are then run against the master files to produce the desired reports, documents, or posting operations. While data is accumulating for one aspect of an operation, another operation isbeing performed on the computer.

As contrasted with the in-line computer, the off-line computer cannot be used economically on a demand basis. This is primarily because the media for storing the large volume master files of a business operation are generally magnetic tapes. Access to information on these tapes must be accomplished in a sequential fashion rather than in a random fashion as would be required if the computer is to be used on demand. This is, of course, not a limitation of flexibility of the computer, but is simply a matter of economics.

The electronic computer performs automatically all of the five basic data processing operations previously discussed. This is achieved by recording on a magnetic tape the sequence of instructions which the computer is to follow in its role as a data processor.

The instructions recognized by the computer permit it to perform: the reading of documents after they have been transcribed onto a magnetic tape, the writing of documents or reports, sorting and classifying information, calculating and exercising simple decisions as required.

The off-line computer incorporates the advantages of the in-line system in performing its operations at high-speed with a great degree of accuracy and at low unit cost for high volume processing; but it eliminates one disadvantage in that file contraction and expansion is simply a matter of the amount of magnetic tape used (a much more economical storage medium), and its flexibility is such that all aspects of a data processing area can be automated. The fact that a batching operation is required for efficient use (and that completely current files are thereby precluded) is meliorated by the speed at which operations are performed -- so that, in general, files are more current than would be possible with any other data processing system except an in-line computer relegated exclusively to one application.

In contrasting in-line equipment with off-line equipment, it has been implied that existing in-line equipment is most often "special purpose". The Univac File-Computer is an example of one computer which is not "special purpose" and yet may be used in line. The term "general purpose" is applied to systems which can be applied to any data processing area for which the computer is given appropriate instructions. General purpose computers are characterized by having stored programs of alterable instructions.

ANALOGUE AND DIGITAL COMPUTERS

There are two methods of measuring quantities, and two types of computers designed around the two methods. Quantities are represented with numbers or with comparisons to the quantity to be represented. A man says a fish is 12 inches long or indicates its length with the distance between extended palms. Computers which count with numbers are called digital computers and those which operate without counting, but derive their results directly from the magnitudes of electric currents, voltages, or shaft rotations, are called analogue computers. Digital computers, exclusively, are used for data processing. These systems employ an individual code for each number and for each letter of the alphabet, and often for punctation marks, for abbreviations such as $, %, & etc., and other symbols.

COMMERCIAL AND SCIENTIFIC COMPUTERS

While the Univac II Data Automation System is frequently employed for the solution of scientific problems, its more common use is for business applications such as the computation of large payrolls, inventories, etc. These problems are characterized by the vast quantity of input and output data. Very few calculations must be performed to compute each individual paycheck; the problem lies in the number of pay checks to be computed. The name "data processing" is applied to these problems.

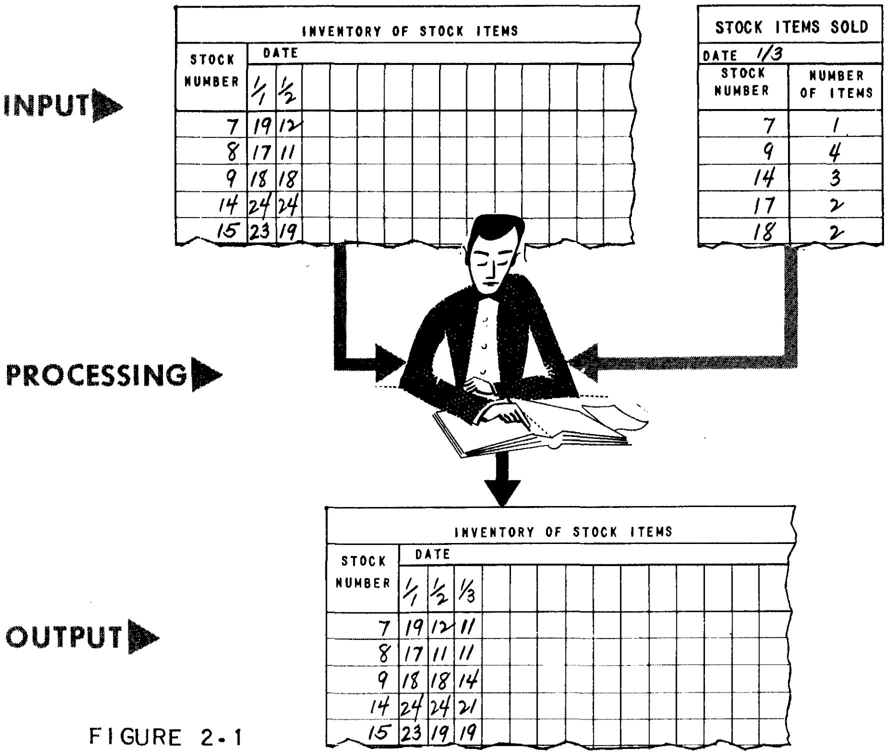
Scientific problems, on the other hand, involve small amounts of information upon which a vast number of operations must be performed. Because of this difference in the nature of business and scientific problems, there are computers designed for emphasis in each field.

The Univac II Central Computer is a general purpose, digital computer used offline primarily for data processing applications.

chapter 2

Elements of the Univac Data Automation System

To determine the elements of a data processing system, examine the steps in the manual solution of a data processing application. Consider a company that·keeps a record of its stock in a ledger. Each day a clerk is supplied with a sales form. On the basis of the form the clerk brings the inventory up to date by writing a new column in the ledger.



INVENTORY OF STOCK ITEMS

STOCK NUMBER	DATE 1/1	1/2
7	19	12
8	17	11
9	18	18
14	24	24
15	23	19

STOCK ITEMS SOLD

DATE 1/3

STOCK NUMBER	NUMBER OF ITEMS
7	1
9	4
14	3
17	2
18	2

INVENTORY OF STOCK ITEMS

STOCK NUMBER	DATE 1/1	1/2	1/3
7	19	12	11
8	17	11	11
9	18	18	14
14	24	24	21
15	23	19	19

FIGURE 2-1

To do the processing the man goes through certain steps.

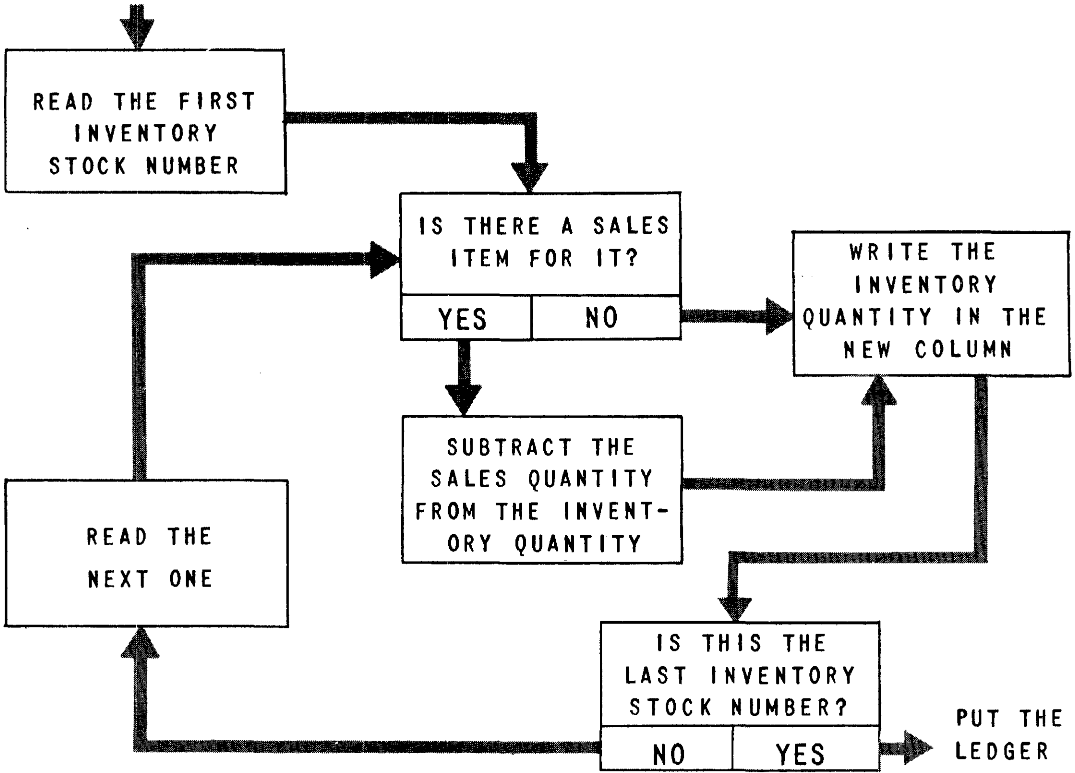

READ THE FIRST INVENTORY STOCK NUMBER
IS THERE A SALES ITEM FOR IT?
YES
NO
WRITE THE INVENTORY QUANTITY IN THE NEW COLUMN
SUBTRACT THE SALES QUANTITY FROM THE INVENT-ORY QUANTITY
READ THE NEXT ONE
IS THIS THE LAST INVENTORY STOCK NUMBER?
NO
YES
PUT THE LEDGER AWAY

FIGURE 2-2

Thus, the clerk must be able to perform arithmetic;

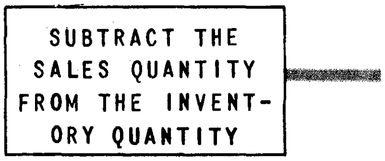


FIGURE 2-3

he must be able to make logical decisions;

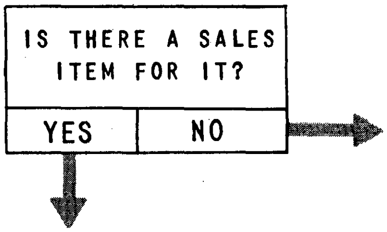


FIGURE 2-4

he must be able to remember information;

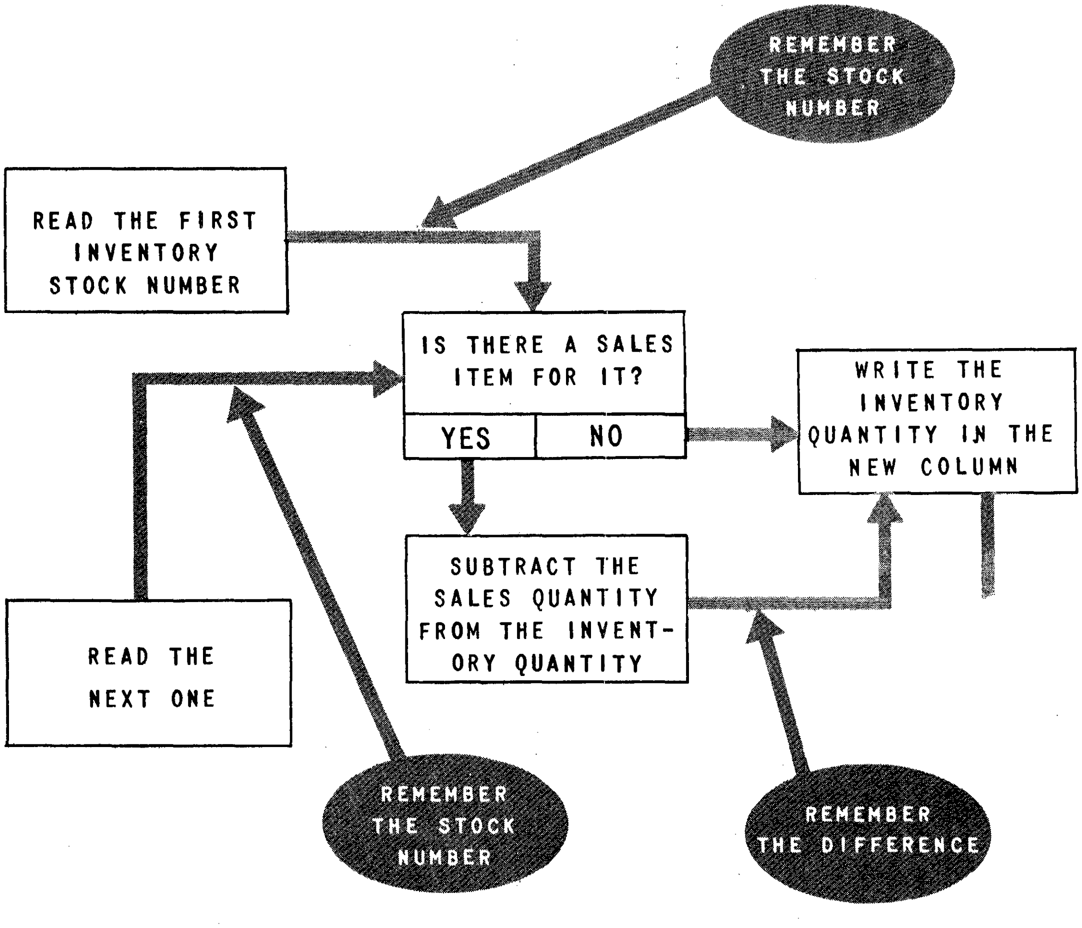

REMEMBER THE STOCK NUMBER
READ THE FIRST INVENTORY STOCK NUMBER
IS THERE A SALES ITEM FOR IT?
YES
NO
WRITE THE INVENTORY QUANTITY IN THE NEW COLUMN
SUBTRACT THE SALES QUANTITY FROM THE INVENTORY QUANTITY
READ THE NEXT ONE
REMEMBER THE STOCK NUMBER
REMEMBER THE DIFFERENCE

FIGURE 2-5

and he must either execute the steps in the sequence shown or do something logically equivalent to this sequence of steps.

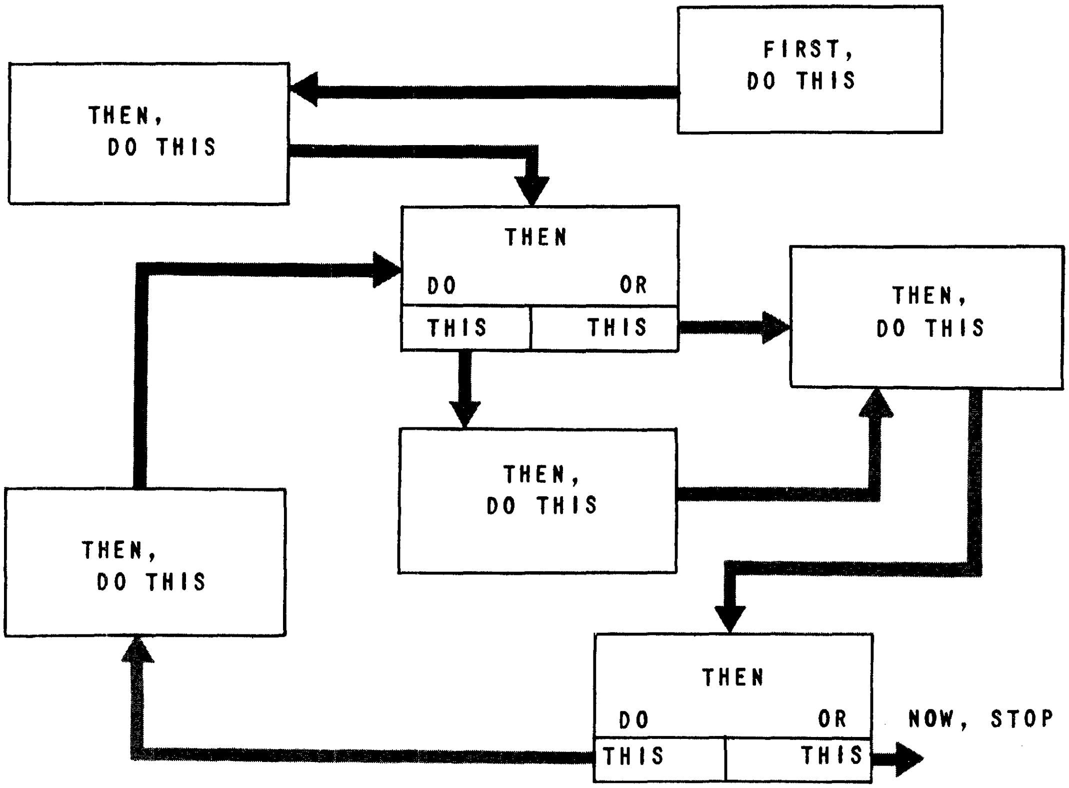
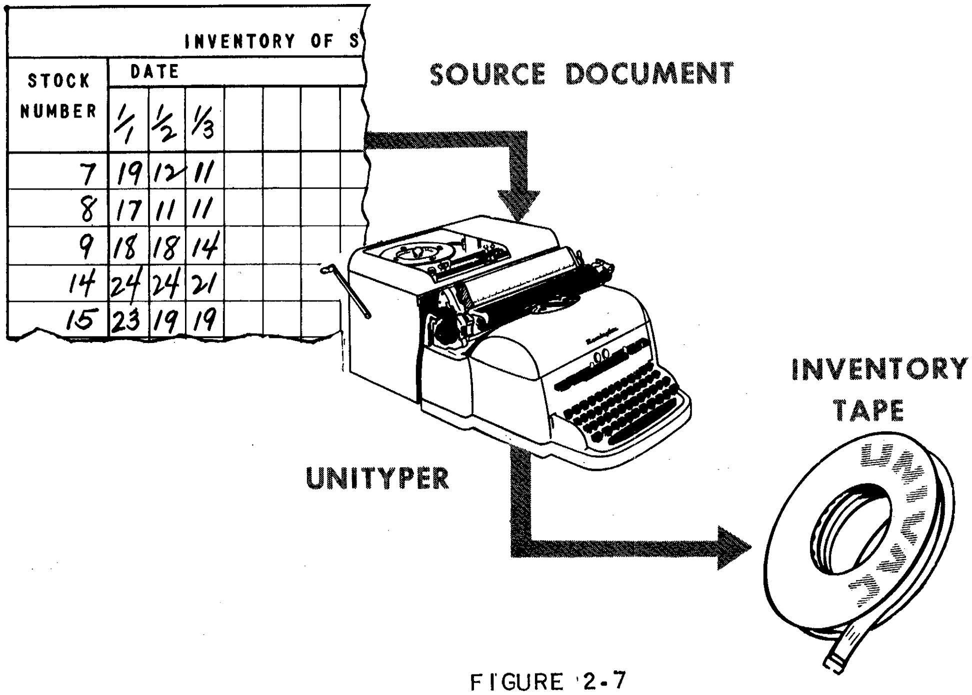


FIGURE 2-6

This example involves six elements.

1 Input
2 Arithmetic
3 Logical Decisions
4 Memory
5 Control
6 Output

Contrasted to the manual system, the Univac Data Automation System keeps the inventory recorded on magnetic tape. Initially the tape would have been prepared by means of the Univac Unityper, a modified typewriter that produces, in addition to typewritten copy, the recorded tape.



INVENTORY OF S						
STOCK NUMBER	DATE					
	1/1	1/2	1/3			
7	19	12	11			
8	17	11	11			
9	18	18	14			
14	24	24	21			
15	23	19	19			

FIGURE 2-7

Instead of a sales form, a sales tape is produced daily, also by the Unityper. Instead of the clerk, the Univac Central Computer does the processing.

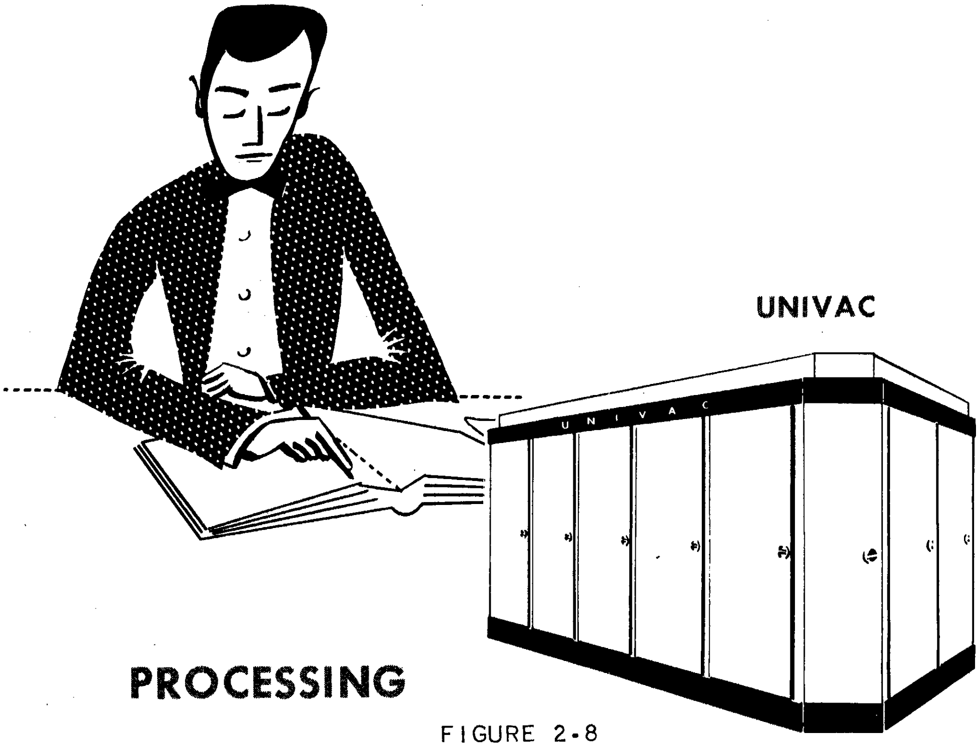


FIGURE 2-8

The inventory tape is read by means of a tape handling mechanism called a Uniservo.

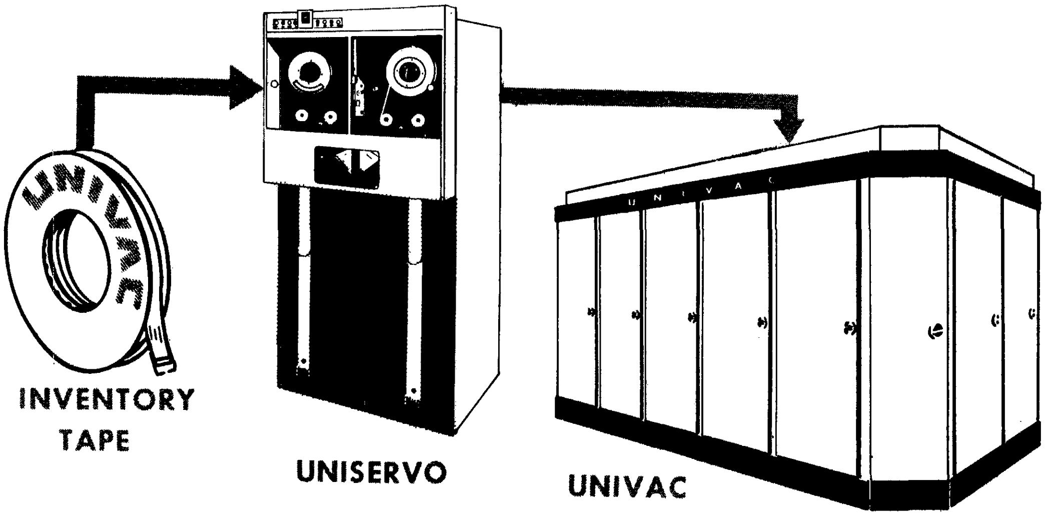


FIGURE 2-9

The sales tape is read from another Uniservo.

The clerk brought the inventory up to date by writing a new column in the ledger. The Central Computer brings the inventory up to date by writing an updated inventory tape on a third Uniservo.

In this application the Central Computer requires three Uniservos - two for reading and one for writing. Reading and writing requirements vary from application to application. To provide maximum flexibility, the Central Computer has access to a bank of 16 Uniservos, any of which can be used for reading or writing.

In the manual solution, the column the clerk writes in the ledger on any one day, that is, the inventory output, becomes the inventory input on the next day. The sales form continues to originate each day from outside the data processing system.

Similarly, in the Univac System, the updated inventory tape written one day becomes the next day's inventory tape, while the sales tape continues to originate each day from outside the system. Once the inventory tape has initially been unityped it need never be unityped again, since it is kept up to date by the Central Computer.

DATA PROCESSING SYSTEM

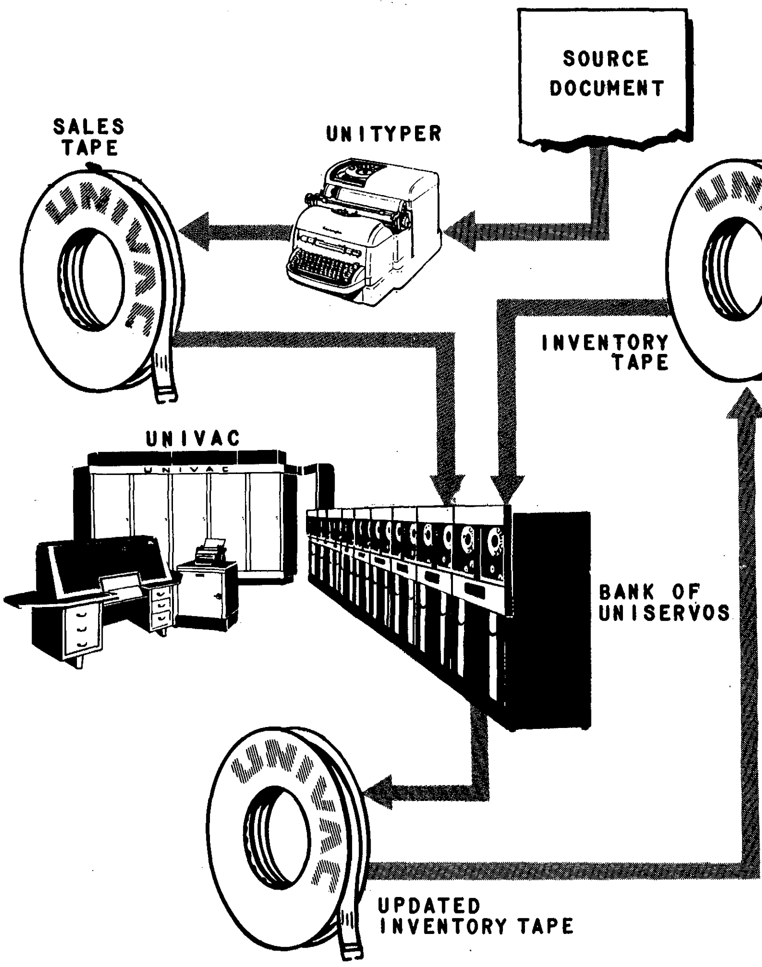


FIGURE 2-10

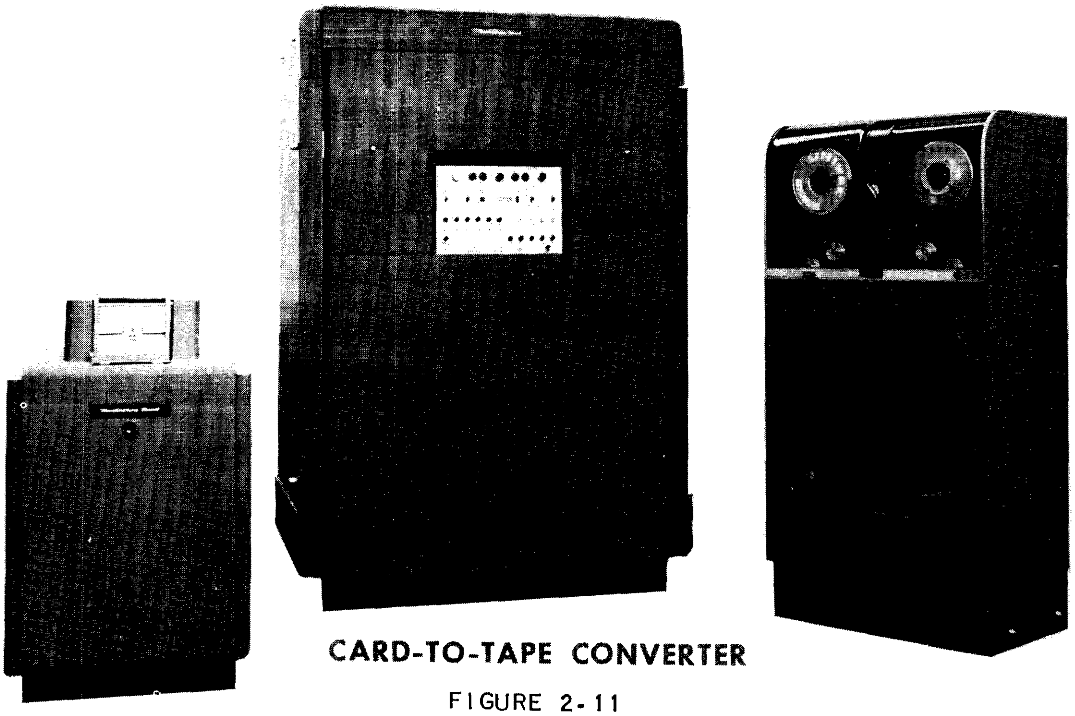
INPUT OUTPUT UNITS

In many cases, input data does not come, and output data is not desired, in tape form. The Univac Data Automation System includes several input units to convert data from some other form to tape, and output units to convert tape data to some other form.

INPUT UNITS

The Unityper has already been discussed as an input unit.

The Univac Card-to-Tape Converter converts data punched on cards to tape.



CARD-TO-TAPE CONVERTER

FIGURE 2-11

The Univac PTM converts data punched on paper tape to magnetic tape.

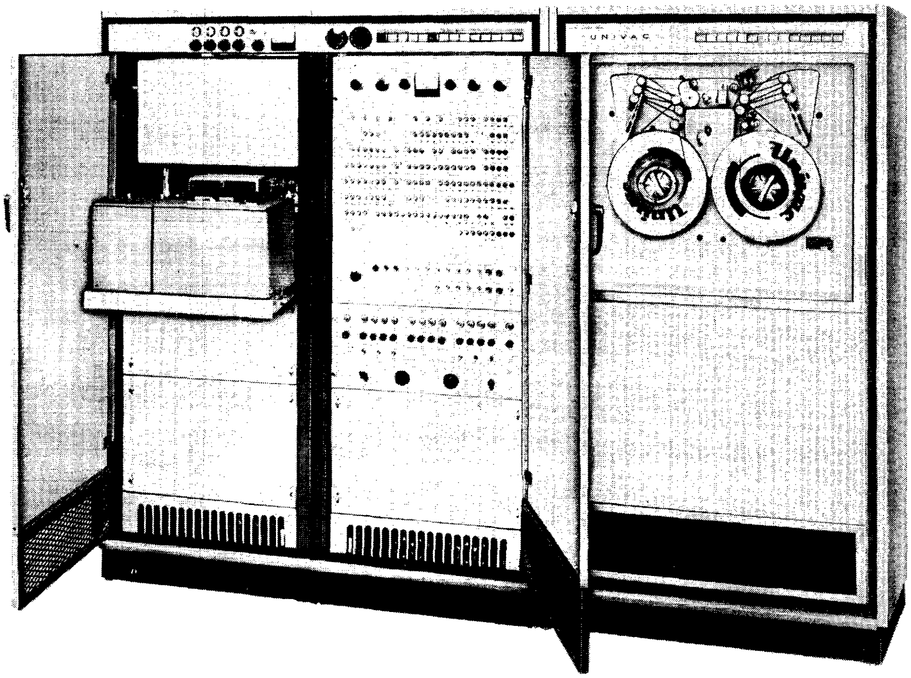


FIGURE 2-12 PAPER-TO-MAGNETIC TAPE CONVERTER

OUTPUT UNITS

The Univac High-Speed Printer.

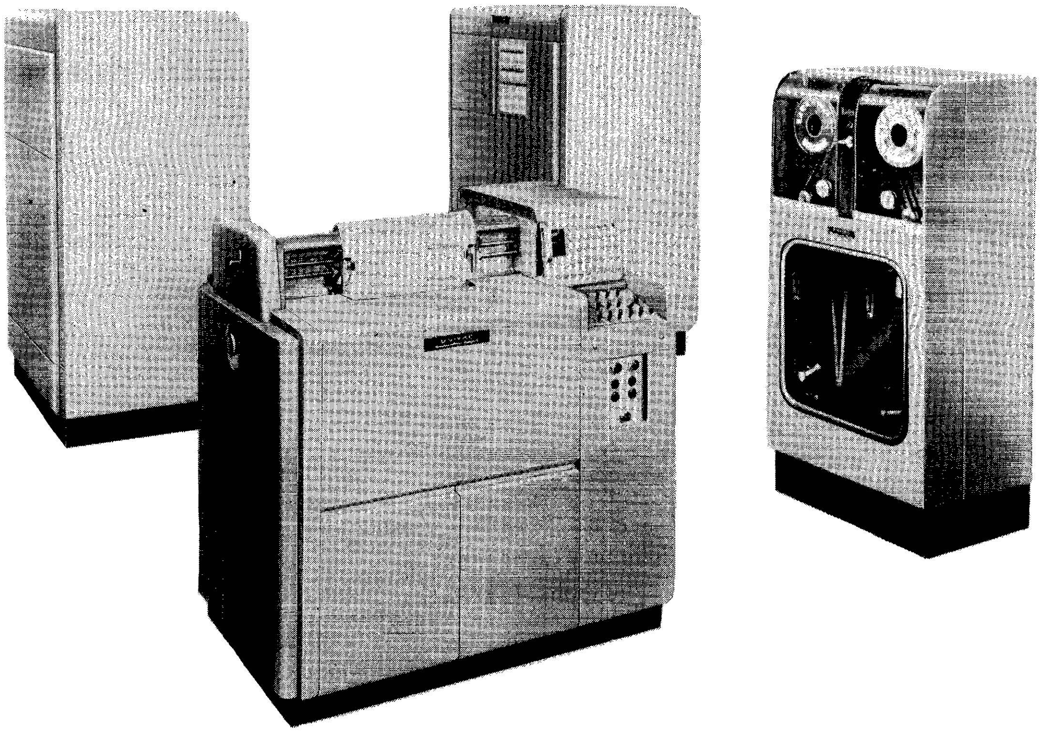


FIGURE 2-13 **HIGH-SPEED PRINTER**

The Univac Tape-to-Card Converter.

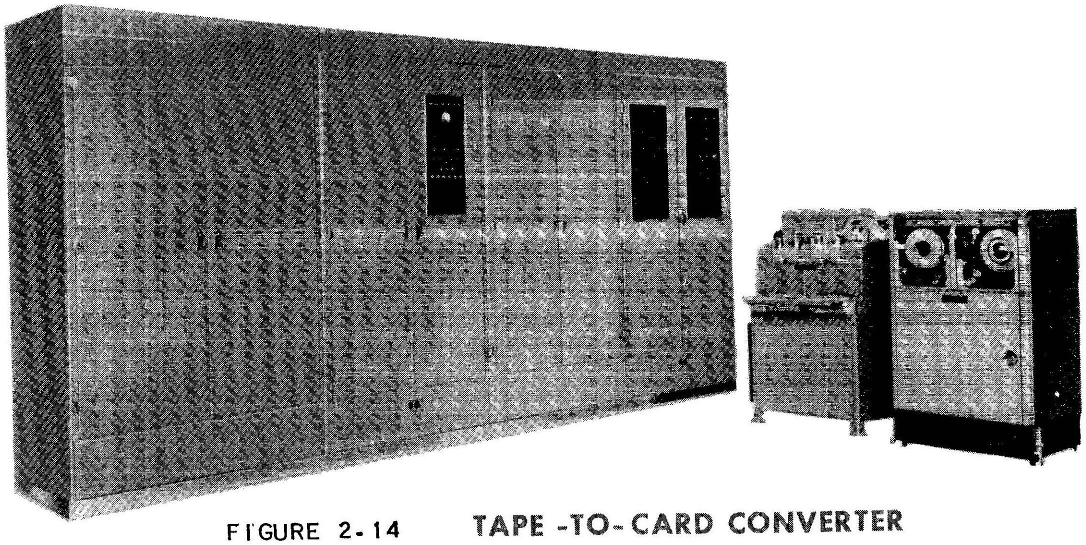


FIGURE 2-14 **TAPE -TO- CARD CONVERTER**

The Univac MTP converts magnetic to paper tape.

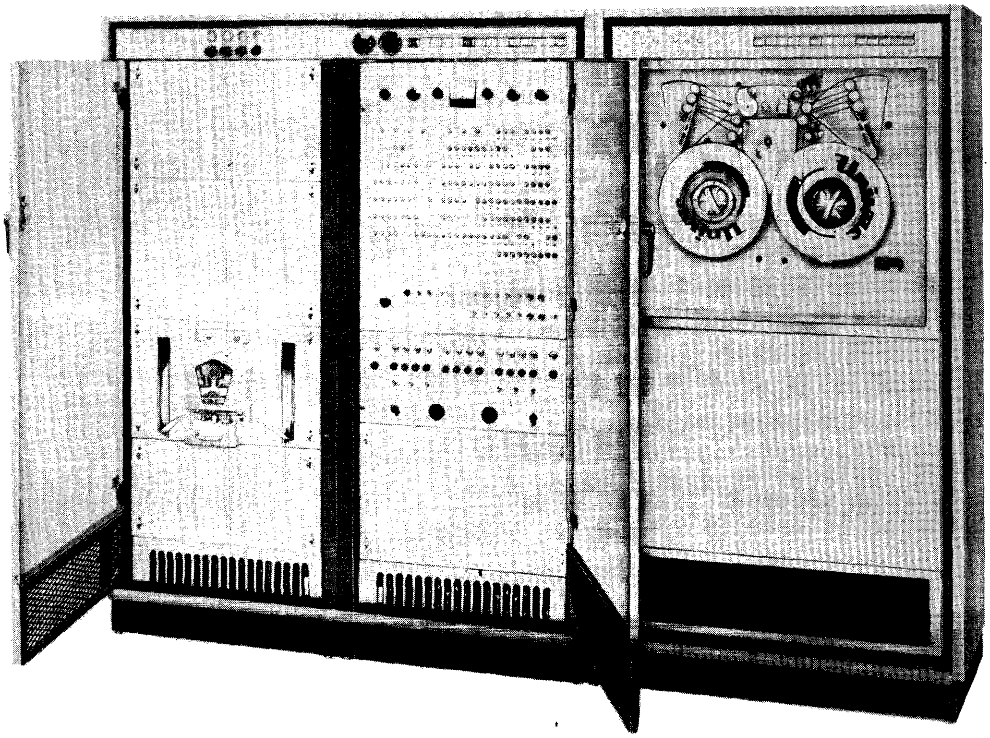


FIGURE 2-15 **MAGNETIC-TO-PAPER TAPE CONVERTER**

KEYBOARD INPUT OUTPUT

Besides using tape, the Central Computer can also accept and produce small volume data directly by means of a keyboard and a typewriter.

The Central Computer accepts data directly from an operator's key strokes on the Supervisory Control Keyboard.

The Central Computer produces printed data directly on the Supervisory Control Printer, which is a modified typewriter.

THE UNIVAC CENTRAL COMPUTER

To satisfy the requirements of an automatic data processor, the Univac Data Automation System must not only be able to accept input and produce output, but must also incorporate the other functions of a data processor, memory, control, arithmetic and logical decision.

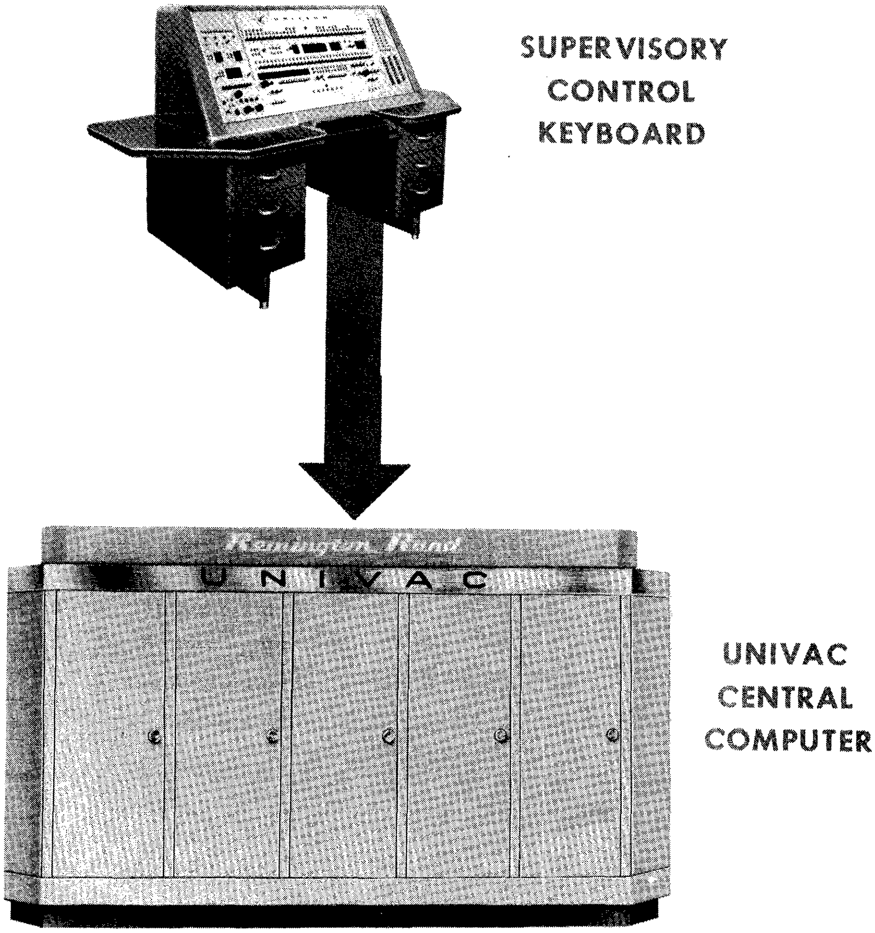


FIGURE 2-16

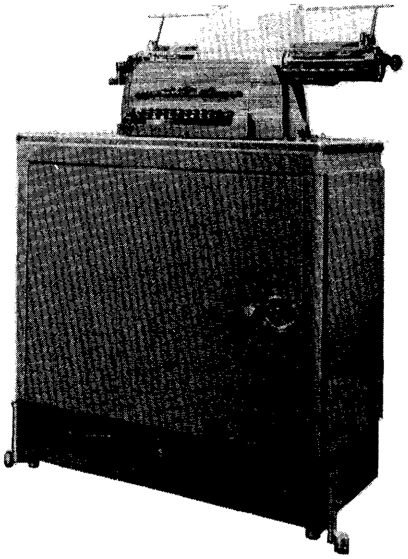


FIGURE 2-17
SUPERVISORY CONTROL PRINTER

These functions are performed by the Central Computer of the Univac System. The memory function is performed by the Central Computer's memory unit; the control function, by the Central Computer's control unit; and the arithmetic and logical decision functions, by the arithmetic unit.

THE MEMORY UNIT

In the manual system described above, all information necessary to the processing is made available to the clerk in some form.

1. The stock number and inventory and sales quantities are on the ledger page and sales form.

2. The date of the current updating is on a calendar.

3. The instructions for updating the inventory are in a procedures manual.

The above information can be classified as:

1. data,

2. constants,

3. instructions.

Similarly, in the Univac System, all necessary information is made available to the Central Computer; the data, on an input tape; the constants and instructions, on an instruction tape.

However, to have the information available is not sufficient for the clerk to do the processing. While processing, the clerk must remember the information bearing on the current processing step. Moreover, the clerk must remember the results of any calculation done at least until he writes the results in the ledger. Similarly the Central Computer must "remember" the data, constants and instructions that it reads from tape, and must "remember" the results of calculations until it writes them on the output tape. The Central Computer "remembers", or stores, information in its memory unit. The memory is divided into cells. Any cell can be used to store data, constants or instructions. The 63 characters used to represent information are shown below.

i	r	t	Σ
Δ	,	"	β
-	.	|	:
0	;	)	+
1	A	J	/
2	B	K	S
3	C	L	T
4	D	M	U
5	E	N	V
6	F	O	W
7	G	P	X
8	H	Q	Y
9	I	R	Z
'	#	$	%
&	¢	*	=
(	@	?	NOT USED

FIGURE 2-18

CHARACTERS

One cell can store one "word", a word being any permutation of twelve characters. The following are examples of words.

JOHNΔJΔJONES
JUNEΔ10Δ1926
012345678901
A00100C00200

The positions of the characters in a word are named as follows.

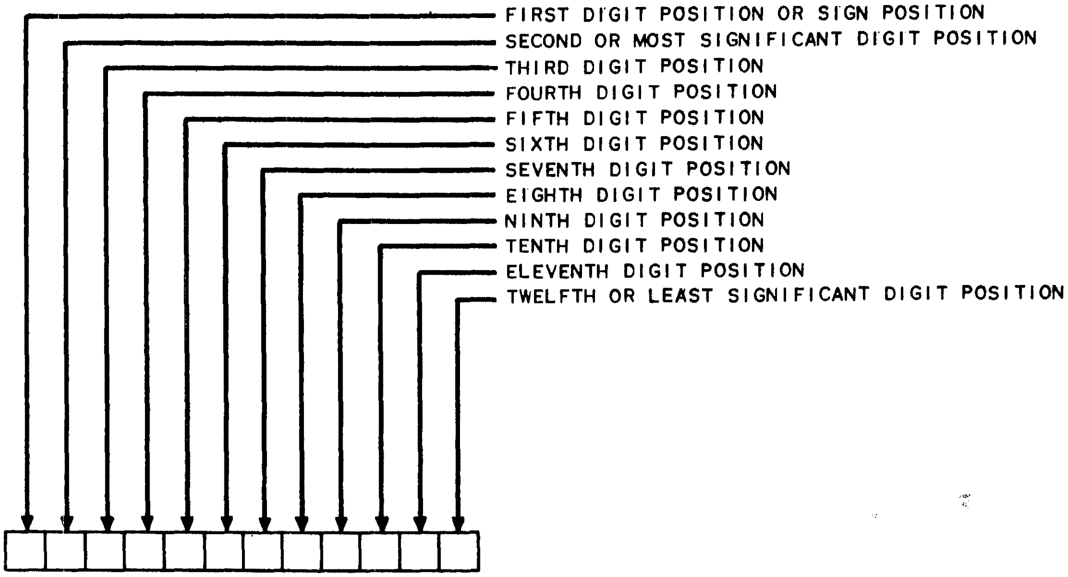


FIGURE 2-19

If a word represents an algebraic quantity, the sign of the quantity must be in the sign position. A plus sign is represented by a zero; a minus sign, by a minus.

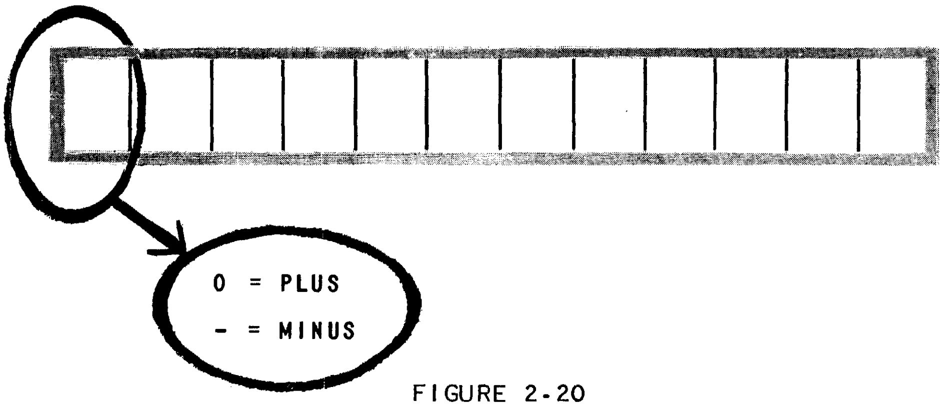


FIGURE 2-20

WORD AS A SIGNED QUANTITY

The basic memory size is 2000 cells, with a 10,000 cell memory available. For the purpose of referring to words in the memory, each cell is given a distinct address. A word in the memory is distinguished from all other words in the memory by the address of the cell in which it is stored. The cells are addressed consecutively from 0000 to 1999. For the 10,000 cell memory the enumeration is continued to 9999.

Once a word has been transferred to a cell, it remains in that cell until another word is transferred to take its place.

Figure 2-21 is a stylized version of the memory unit storing instructions, data and constants.

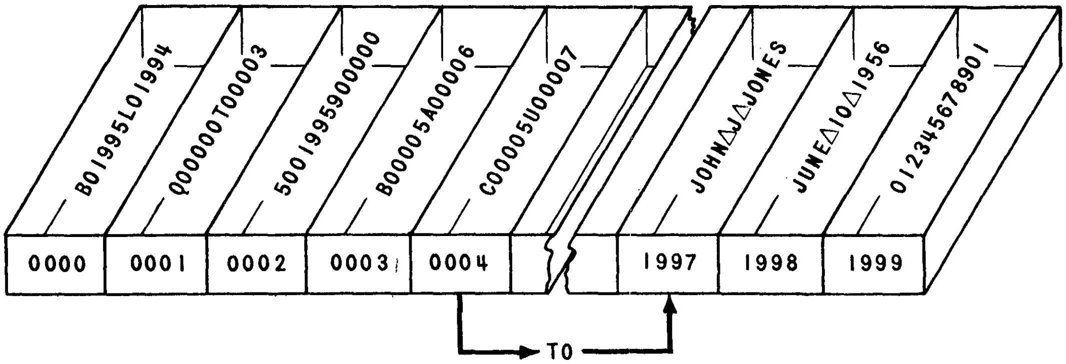
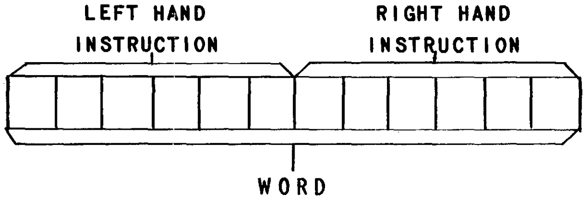


FIGURE 2-21

THE CONTROL UNIT

The code for an instruction is represented in six characters. Consequently, two instructions, called an instruction pair, are represented in one word.

FIGURE 2-22
INSTRUCTION PAIR

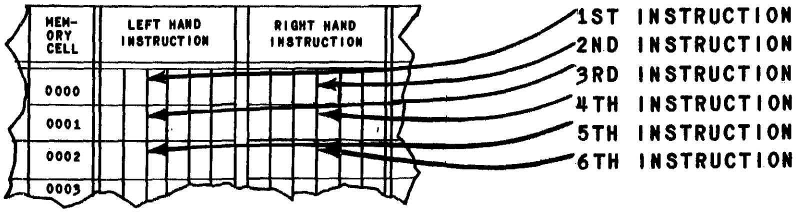


The function of the control unit is to select, in the proper sequence, each instruction in the memory, interpret it and execute it. Instructions are selected in pairs, one word, at a time. The left hand instruction (LHI) is executed, and then the right hand instruction (RHI). Thus, the control unit operates on a three stage cycle.

1. Select an instruction pair from the memory.
2. Execute the LHI.
3. Execute the RHI.

The selection of instruction pairs is performed in a sequential manner. That is, if the instruction pair just executed is in cell 0019, the next pair to be executed is in cell 0020.

Initially the control unit begins the sequential execution of instruction pairs with the pair in cell 0000. Thus, to have instructions executed in sequence, it is only necessary to represent the first instruction in the LHI of the word in cell 0000; the second in the RHI of the word in cell 0000; the third in the LHI of cell 0001; and so on.



CONTROL

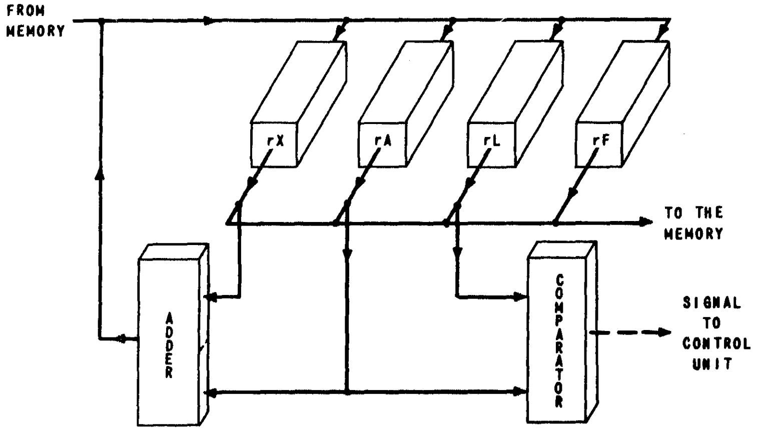
FIGURE 2-23

THE ARITHMETIC UNIT

The arithmetic unit has characteristics in common with a desk calculator in that it contains an adder to produce the sum or difference of two words, a multiplier the product, and a divider to produce their quotient. In addition, to enable the Central Computer to make logical decisions, the arithmetic unit contains a comparator, which inspects two words to determine their equality or relative magnitude.

To operate on a word in the memory, the Central Computer must transfer the word to the arithmetic unit. To provide storage for such words, the arithmetic unit contains four registers named A, X, L and F. The arithmetic registers are identical to memory cells except that they are auxiliary to the memory. The registers serve the arithmetic unit in the same way as dials serve a calculator; each register storing either a word to be operated on or the result of an operation.

Figure 2-24 is a stylized version of a portion of the arithmetic unit.



ARITHMETIC UNIT

FIGURE 2-24

The memory, control and arithmetic units and their interrelations are shown here: (The 60 word registers I and O, used for input and output and the multiword registers W and Z will be described in detail in a later chapter).

chapter 3

Introduction to Coding

The preparation of a problem for its solution by The Univac Data Automation System is called programming. Programming is done in three steps.

1. Process Charting - The layout of the data processing system in terms of input, output and processing.

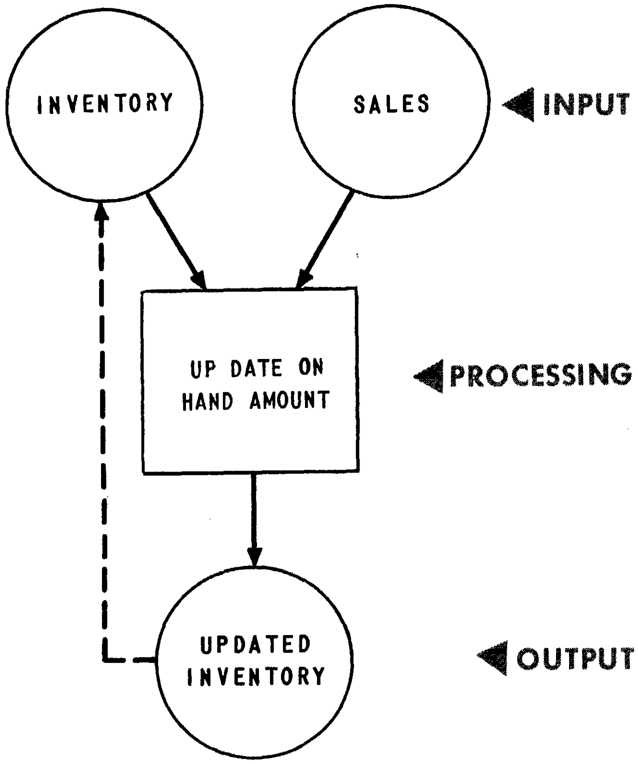
2. Logical Analysis - The analysis of the processing into a sequence of "small" logical steps.

3. Coding - The translation of the logical analysis into instructions.

PROCESS CHARTING

Figure 3-1 is a process chart.

In this manual all problems requiring logical analysis and coding are given in discursive form. All the problems specify three things - input, processing and output and could be put in process chart form which is the usual basis for analysis and coding.



PROCESS CHART

FIGURE 3-1

CODING

Computers usually perform a function in a series of operations. Each operation is executed under the influence of an instruction. An instruction specifies at least two things.

1. the operation to be performed.
2. the data to be operated on.

The data is usually specified in terms of the storage in which the data is to be found. For example, the data might be specified in terms of the address of the cell in which it is stored.

A computer might perform the function of adding two quantities together and recording the sum in three operations.

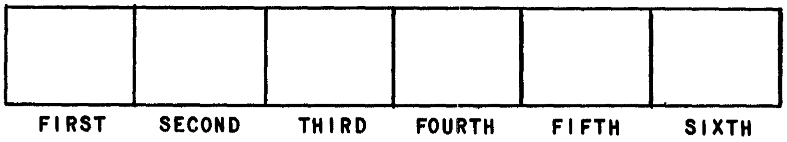
1. Select one quantity.
2. Add the second quantity to the first.
3. Record the sum.

If one quantity is in cell 1880; the other, in 1881; and if the sum is to be stored in cell 1882; the instructions to cause the computer to do the above operations might be:

1. BRING 1880
2. ADD 1881
3. CLEAR 1882

where BRING, ADD and CLEAR are code for the operations to be done; and 1880, 1881 and 1882, the addresses of the cells in which the data is stored.

In the central computer of the Univac Data Automation System an instruction consists of six characters, named as follows.



INSTRUCTION DIGITS

FIGURE 3-2

The first and second instruction digits indicate what operation is to be performed; the third through sixth digits, the address of the word affected by the operation.

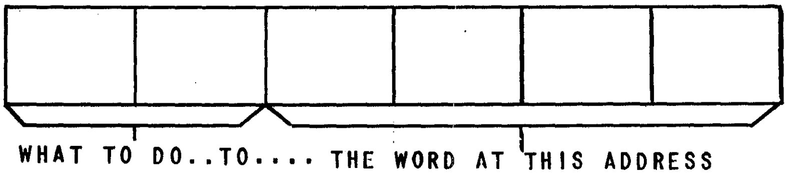


FIGURE 3-3

The instruction

501880

tells the central computer to perform the operation indicated by "50" on the word in cell 1880.

ARITHMETIC INSTRUCTIONS - LIST A

An "m" is used to symbolized the third through sixth instruction digits. Parentheses are used to symbolize "the contents of". The symbol

(m)

means "the contents of cell m". An "r" is used to symbolize "register". The symbol

$$rA$$

means "register A". An arrow is used to symbolize "is (are) transferred to". The symbol

$$(m) \rightarrow rA$$

means "(m) are transferred to rA".

To process data, the computer must read the data from tape and store it in the memory. There are instructions that, when executed, do the reading. These instructions will not be discussed at this time. Instead, reading data will be indicated by the words, "Read Data".

INSTRUCTION	OPERATION	MNEMONIC
BOm	$(m) \rightarrow rA, rX$	Bring

Transfer (m) to rA and rX, or bring (m) to rA and rX.

INSTRUCTION	OPERATION	MNEMONIC
COm	$(rA) \rightarrow m;\ 0 \rightarrow rA$	Clear

Transfer (rA) to m. Transfer a word of zeros to rA, or clear rA.

One of the possible uses of these instructions is to transfer a word from one cell to another. If the word in cell 1880 is to be transferred to cell 1881, the sequence of instructions might be

B01880 C01881

INSTRUCTION	OPERATION	MNEMONIC
HOm	$(rA) \rightarrow m$	Hold

Transfer (rA) to m.

The mnemonic is to hold (rA) after the transfer to memory. The HOm instruction differs from the COm instruction only in that (rA) remains unchanged.

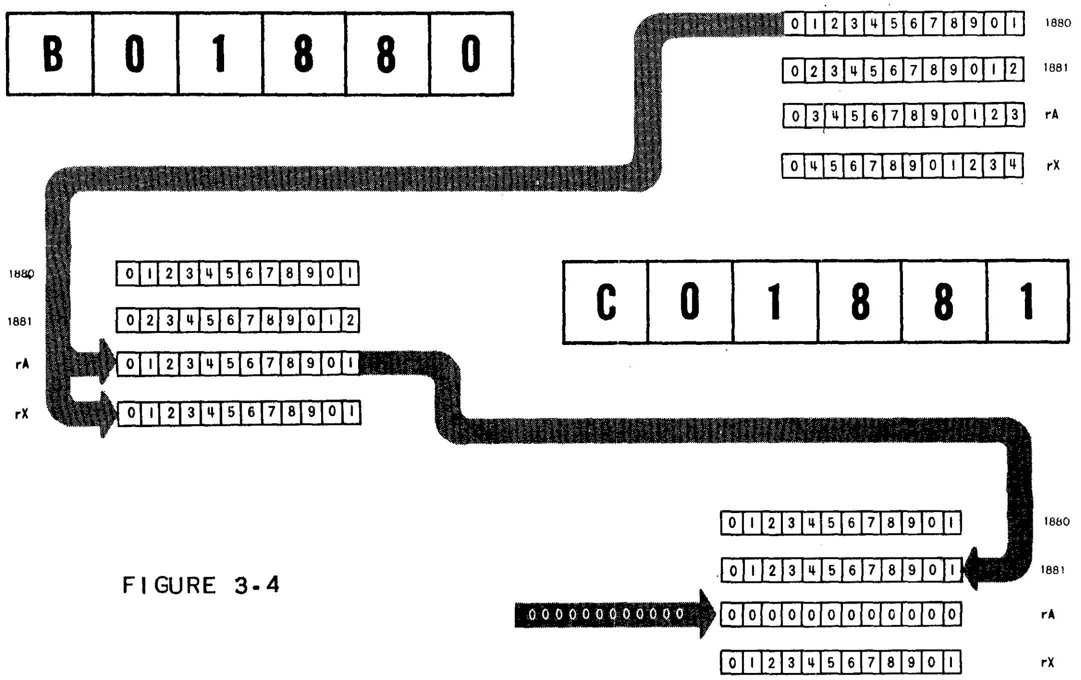


FIGURE 3-4

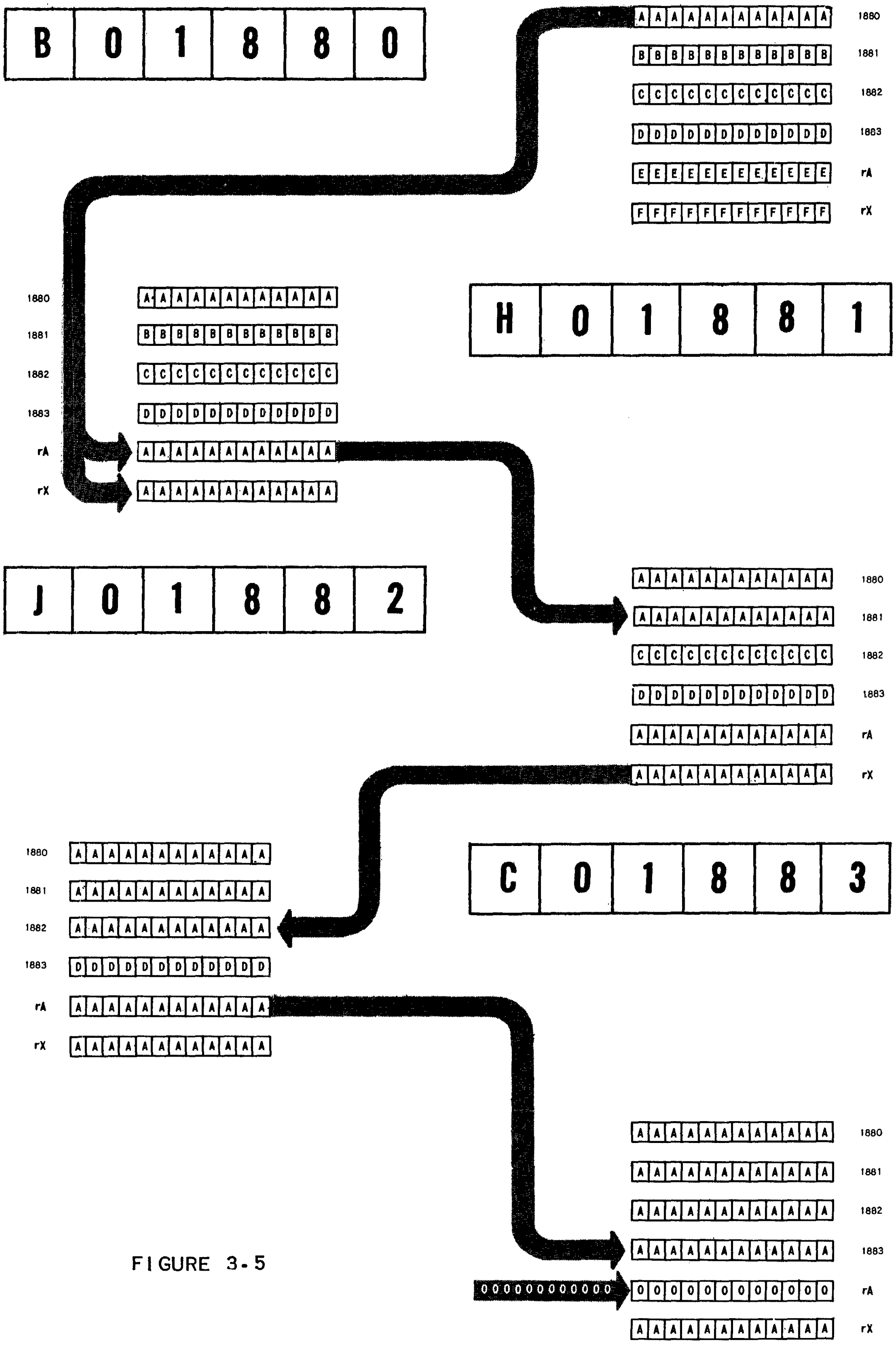
INSTRUCTION	OPERATION
JOm	(rX)⟶m Transfer (rX) to m.

One of the possible uses of these instructions is to duplicate the contents of a certain cell in several other cells. If the contents of cell 1880 are to be duplicated in cells 1881, 1882 and 1883, the instructions might be

```
     B01880  H01881
     J01882  C01883

or:  B01880  J01881
     J01882  J01883

or:  B01880  H01881
     H01882  H01883

etc.
```

INSTRUCTION	OPERATION	MNEMONIC
AOm	(m)⟶rX; (rA) + (rX)⟶rA	Add

Add (rA) and (m), and transfer the sum to rA.



B 0 1 8 8 0
A A A A A A A A A A A A 1880
B B B B B B B B B B B B 1881
C C C C C C C C C C C C 1882
D D D D D D D D D D D D 1883
E E E E E E E E E E E E rA
F F F F F F F F F F F F rX
1880 A A A A A A A A A A A A
1881 B B B B B B B B B B B B
1882 C C C C C C C C C C C C
1883 D D D D D D D D D D D D
rA A A A A A A A A A A A A
rX A A A A A A A A A A A A
H 0 1 8 8 1
J 0 1 8 8 2
A A A A A A A A A A A A 1880
A A A A A A A A A A A A 1881
C C C C C C C C C C C C 1882
D D D D D D D D D D D D 1883
A A A A A A A A A A A A rA
A A A A A A A A A A A A rX
1880 A A A A A A A A A A A A
1881 A A A A A A A A A A A A
1882 A A A A A A A A A A A A
1883 D D D D D D D D D D D D
rA A A A A A A A A A A A A
rX A A A A A A A A A A A A
C 0 1 8 8 3
A A A A A A A A A A A A 1880
A A A A A A A A A A A A 1881
A A A A A A A A A A A A 1882
A A A A A A A A A A A A 1883
000000000000
0 0 0 0 0 0 0 0 0 0 0 0 rA
A A A A A A A A A A A A rX

FIGURE 3-5

The mnemonic is to add (rA) and (m). The computer executes the AOm instruction as follows. (m) are transferred to rX. (rA) and (rX) are added. The sum is transferred to rA.

To add the contents of cell 1880 to the contents of cell 1881 and store the sum in 1882, the sequence of instructions might be

```
B01880 A01881
C01882
```

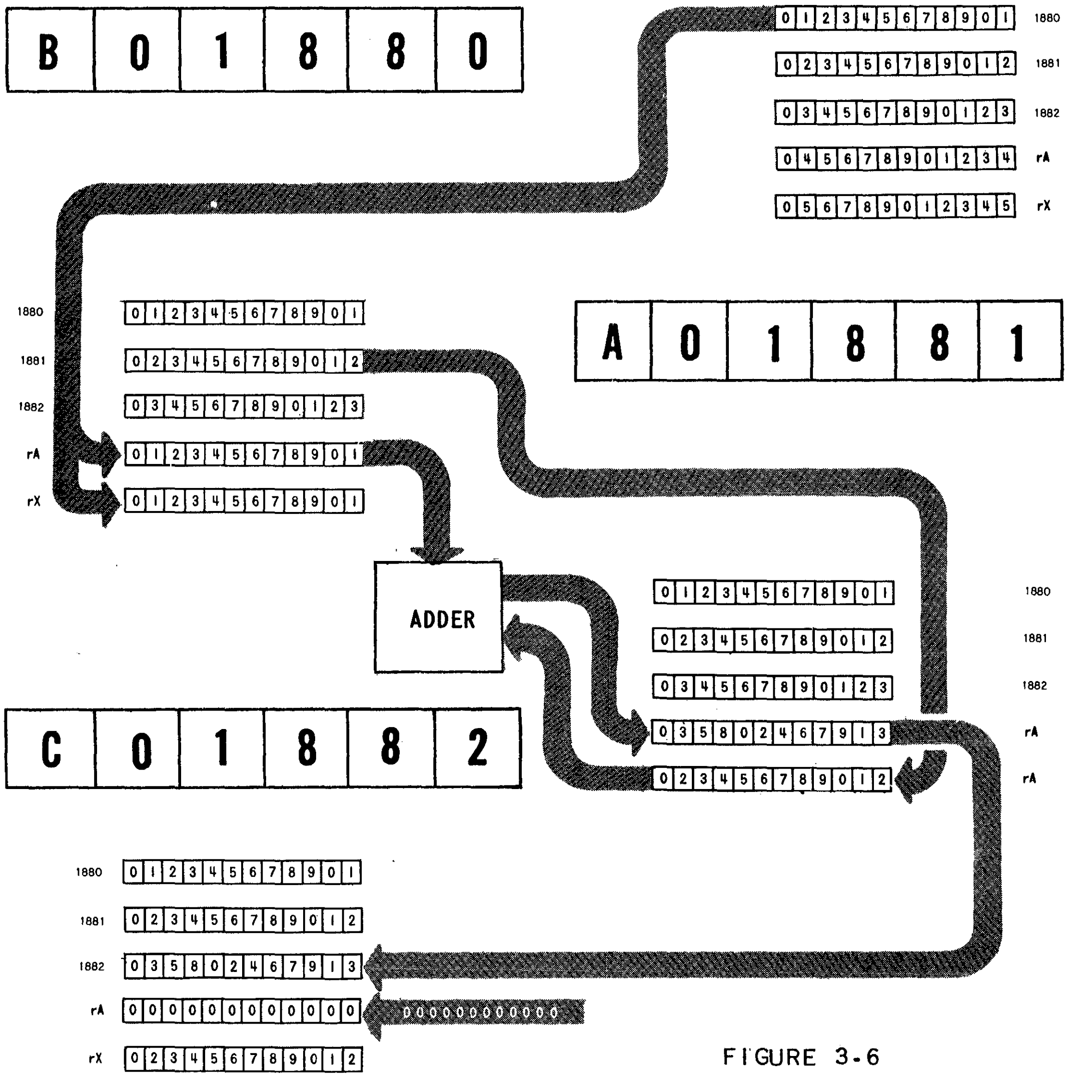


FIGURE 3-6

or

```
B01880 A01881
H01882
```

if it is desired to preserve the sum in rA.

INSTRUCTION	OPERATION
XOm	(rA) + (rX)⟶rA
	Transfer the sum of (rA) and (rX) to rA.

When executing the XOm instruction the computer ignores m.

One of the possible uses of the XOm instruction is to add the same number to a sum more than once. Assuming that a quantity is in cell 1880, the sequence of instructions to build up three times the quantity might be

```
B01880  X00000
X00000
```

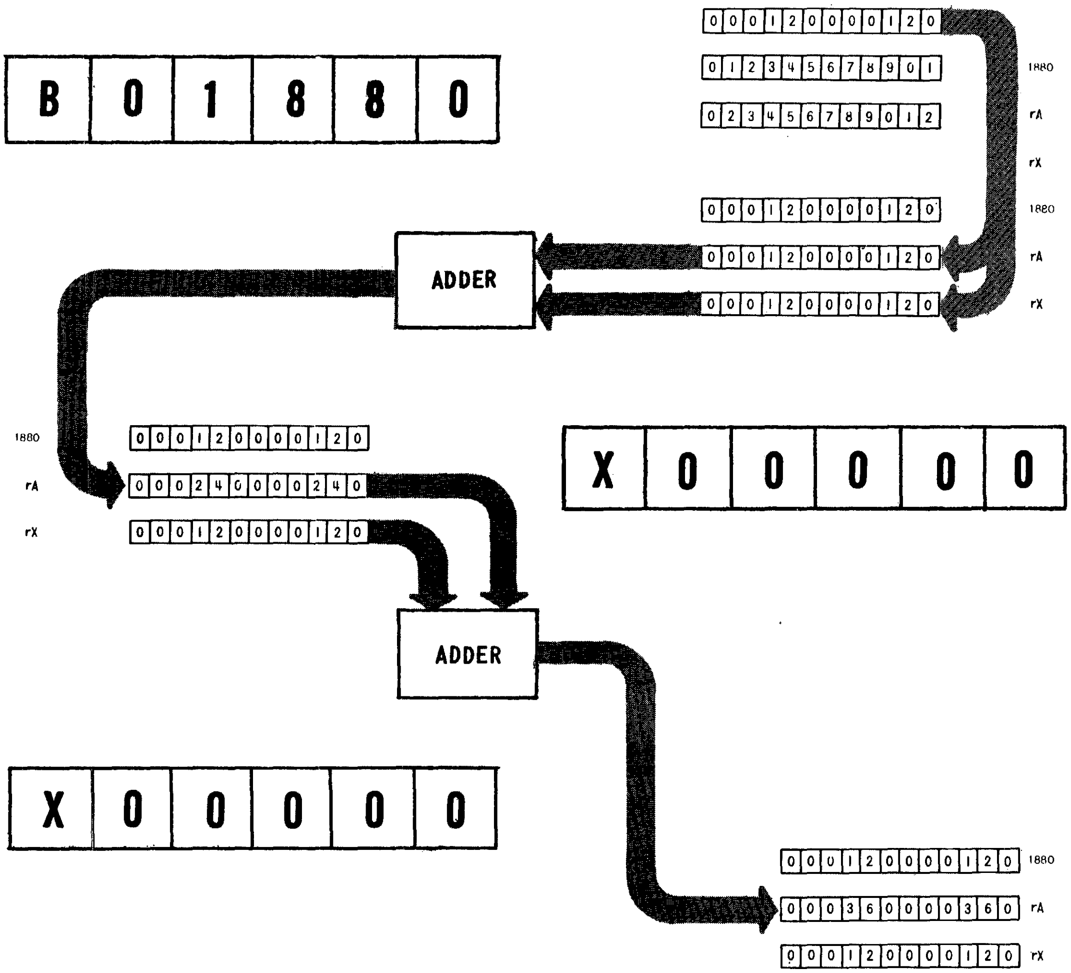
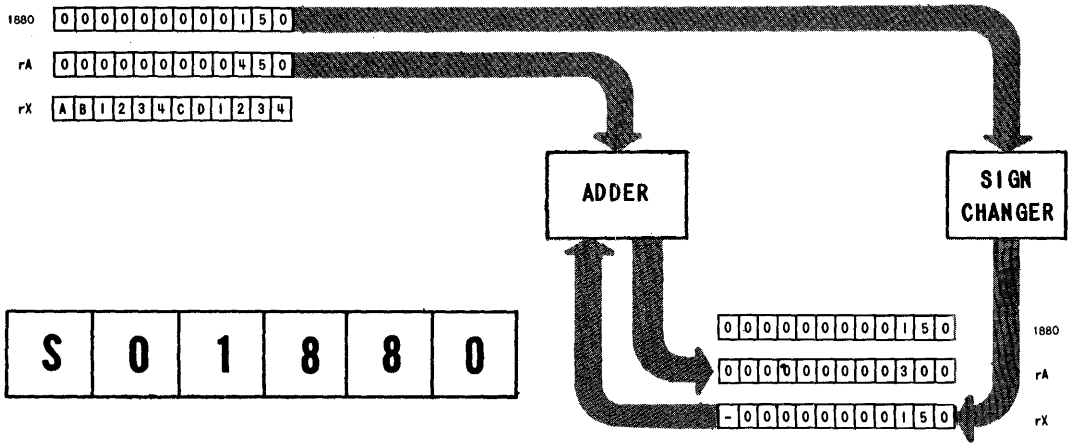


FIGURE 3-7

INSTRUCTION	OPERATION	MNEMONIC
SOm	–(m)→rX; (rA) + (rX)→rA	Subtract

Subtract (m) from (rA). Transfer the difference to rA.

The mnemonic is to subtract (m) from (rA). The computer executes the SOm instruction as follows. Minus the (m) are transferred to rX. (rA) and (rX) are added. The sum is transferred to rA:



1880 0 0 0 0 0 0 0 0 0 1 5 0
rA 0 0 0 0 0 0 0 0 0 4 5 0
rX A B 1 2 3 4 C D 1 2 3 4
ADDER
SIGN CHANGER
S 0 1 8 8 0
0 0 0 0 0 0 0 0 0 1 5 0 1880
0 0 0 0 0 0 0 0 0 3 0 0 rA
- 0 0 0 0 0 0 0 0 1 5 0 rX

FIGURE 3-8

If the contents of a cell are negative, minus the contents would be positive.

INSTRUCTION	OPERATION
50m	(m)→SCP

Print (m) on the Supervisory Control Printer (SCP).

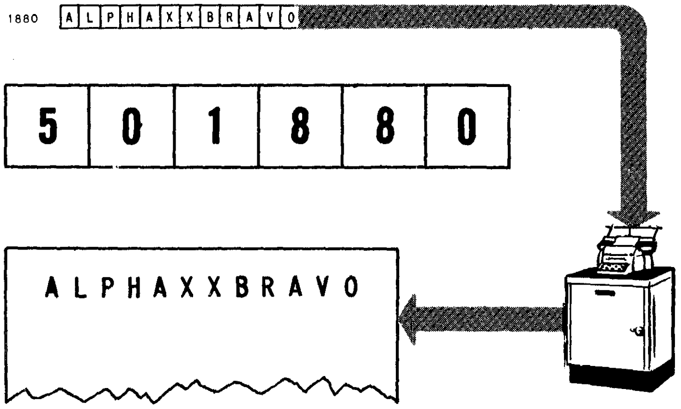


FIGURE 3-9

INSTRUCTION	OPERATION
90m	Stop
	Stop operation

In executing the 90m instruction, the computer ignores m.

ILLUSTRATIVE EXAMPLE:

Reading the data stores the ON HAND quantity of a commodity in cell 1880, the ON ORDER quantity in cell 1881, and the EXPECTED REQUIREMENTS for the next 60 days in cell 1882. Print (on hand) + (on order) − (required). (Data will frequently be stored in memory starting at cell 1880 because of programming convenience. Reasons for this will be described in a later chapter.)

LOGICAL ANALYSIS

1. Read the data.
2. Add the on order to the on hand.
3. Subtract the required from the sum.
4. Print the difference.
5. Stop.

CODING

```
0000     READ                    }   Read the data
                       DATA      }
0001     B01880                  }
                       A01881    }   Add the on order to the on hand
0002     S01882                      Subtract the required from the sum
                       C01883    }
0003     501883                  }   Print the difference
                       900000        Stop
```

The following is a description of the thinking that might have accompanied this coding. (The next new material begins on page 46.)

Since the computer executes instruction pairs by starting with the pair in cell 0000 and moving sequentially through the instruction pairs following, the instruction pairs should be stored in logical sequence, starting in cell 0000. Furthermore, since the computer executes the LHI of an instruction pair before the RHI, the first instruction of a pair to be executed should be coded as the LHI.

The logical analysis shows that the first step is to read the data. This step is shown by writing "Read Data" in cell 0000.

The next step in the analysis is to add the on order quantity to the on hand quantity. The computer will add two quantities if it is given an AOm instruction. But the AOm instruction adds those quantities stored in rA and m. The on hand and on order quantities are in cells 1880 and 1881. Before the quantities can be added together one must be stored in rA. To store a quantity in rA, the BOm instruction can be used. To store the on hand quantity in rA the LHI in cell 0001 should be:

B01880

At the completion of the B01880 instruction the on hand quantity will be in rA. To add the on order quantity to (rA), the instruction needed is

A01881

which should be the RHI of cell 0001.

After the execution of the AOm instruction the computer will have stored the sum of the on hand and on order quantities in rA. The next step is to subtract the required quantity from the sum. This step calls for an SOm instruction where the minuend is in rA and the subtrahend is in the memory. This situation is present, so a S01882 instruction will subtract the required quantity from the sum of the on hand and on order in rA.

The next step is to print the difference. The 50m instruction prints the contents of a cell, but the difference is in rA. Therefore, the contents of rA must be stored in a cell. This storage can be done by means of the COm instruction. The cell specified by the COm instruction must not contain anything necessary to the execution of the remainder of the coding. Cell 1883 meets this requirement, and the instruction could be C01883. The execution of this instruction transfers the difference to cell 1883. Then the execution of the instruction, 501883, will print the difference on SCP.

The last step is to stop operation. The execution of a 90m instruction does this.

It is customary to draw a line under 90m instructions to separate the coding into related segments.

It is also customary to stagger the coding in such a manner that each instruction appears on a separate line. In this way it is possible to identify each step in the logical analysis with certain instructions in the coding. Coding paper has been designed for this purpose.

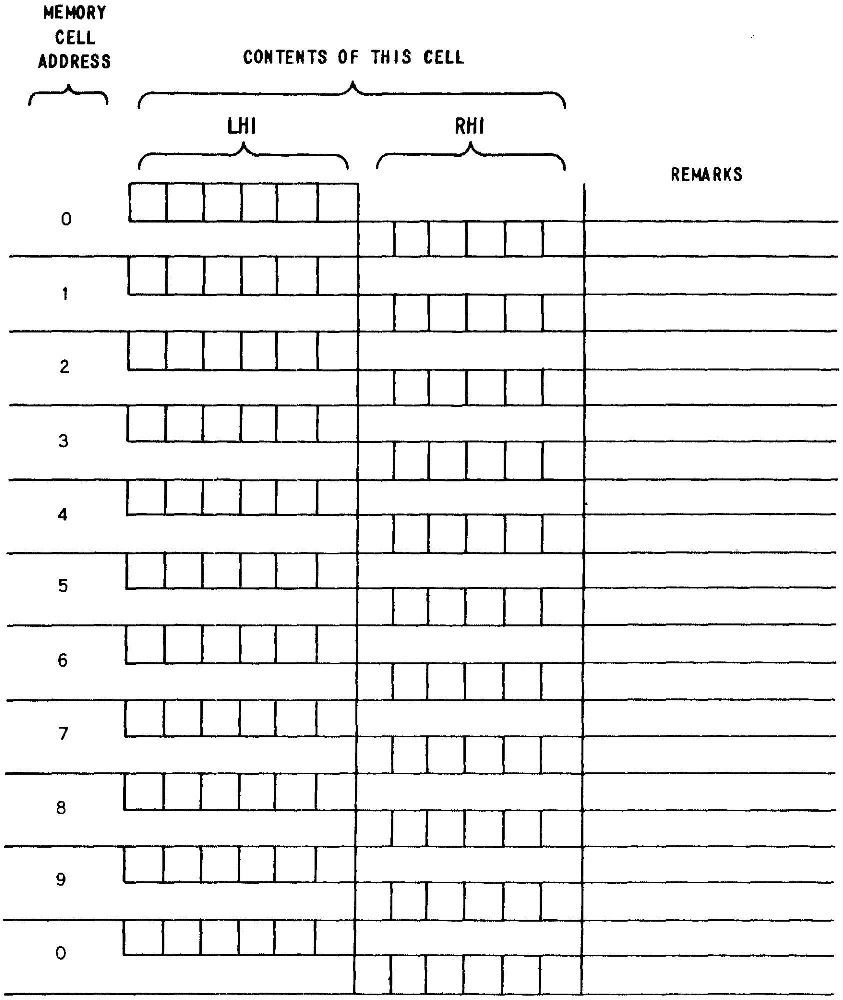


FIGURE 3-10

The left hand side of the paper is for the address of the cell in which an instruction pair is to be stored. The least significant digits of the address are preprinted. Two rows are allotted to each cell,, the upper for the LHI, the lower for the RHI, as indicated by the squares, one square per character. The right hand side of the paper is for remarks.

STUDENT EXERCISES

1. Reading the data stores a quantity in cell 1880. Store the quantity in cells 1881 and 1882.

2. Reading the data stores two quantities in cells 1880 and 1881. Interchange the quantities.

3. Reading the data stores five receipt amounts in cells 1880 - 1884. Print the sum of the receipt amounts.

4. Reading the data stores four quantities, A, B, C and D, in cells 1880-1883. If

$$E = A + B$$
$$F = A + B - C$$
$$G = A + B - C + D$$

print E, F and G.

5. Reading the data stores four quantities, A, B, C, and D, in cells 1880-1883. If

$$R = 2A - B + 3\,(C + D)$$

print R.

ARITHMETIC INSTRUCTIONS - LIST B

INSTRUCTION	OPERATION	MNEMONIC
LOm	(m)→rL., rX	Load

Transfer (m) to rL and rX, or load rL and rX with (m).

INSTRUCTION	OPERATION
KOm	(rA)→rL; 0→rA

Transfer (rA) to rL. Transfer a word of zeros to rA.

In executing the KOm instruction, the computer ignores m.

INSTRUCTION	OPERATION	MNEMONIC
IOm	(rL)→m	Into

Transfer (rL) to m, or transfer (rL) into m.

INSTRUCTION	OPERATION	MNEMONIC
POm	(m)→rX; 3 | (rL) | →rF; (rL) x (rX)→rA [11 MSD], rX [11 LSD]	Precision Multiply

Multiply (rL) by (m). Transfer the 11 most significant digits of the product to rA; the 11 least significant digits to rX.

The execution of the POm instruction produces a precise 22 digit product. The mnemonic is to precision multiply (rL) by (m). The computer executes the POm instruction as follows. (m) are transferred to rX. Three times the absolute value of (rL) are transferred to rF. (The reason for this is described in a later chapter.) (rL) are multiplied by (rX). The 11 most significant digits of the product are transferred to digit positions 2-12 of rA; the 11 least significant digits, to positions 2-12 of rX. The sign of the product is transferred to the sign positions of rA and rX.

INSTRUCTION	OPERATION	MNEMONIC
MOm	(m)→rX; 3 | (rL) | →rF; (rL) x (rX)→rA [11 MSD rounded], rX [11 LSD + .5]	Multiply

Multiply (rL) by (m). Transfer the product to rA.

The execution of the MOm instruction produces an 11 digit rounded product in rA. The mnemonic is to multiply (rL) by (m). The computer executes the MOm instruction in the same way as it executes the POm instruction except that, after the operation associated with the POm instruction is complete, five is added to the most significant digit of (rX), and if a carry is produced, it is added to the least significant digit of (rA).

INSTRUCTION	OPERATION	MNEMONIC
NOm	−(m)→rX; 3| (rL) |→rF; (rL) x (rX)→rA [11 MSD rounded], rX [11 LSD + .5]	Negative Multiply

Multiply (rL) by minus (m). Transfer the product to rA.

The mnemonic is to negative multiply (rL) by (m). The computer executes the NOm instruction as follows. Minus (rX) are transferred to rX. The remainder of the operation is exactly as in the execution of the MOm instruction. The following figure shows the difference in the effect of the execution of the POm, MOm, and NOm instructions.

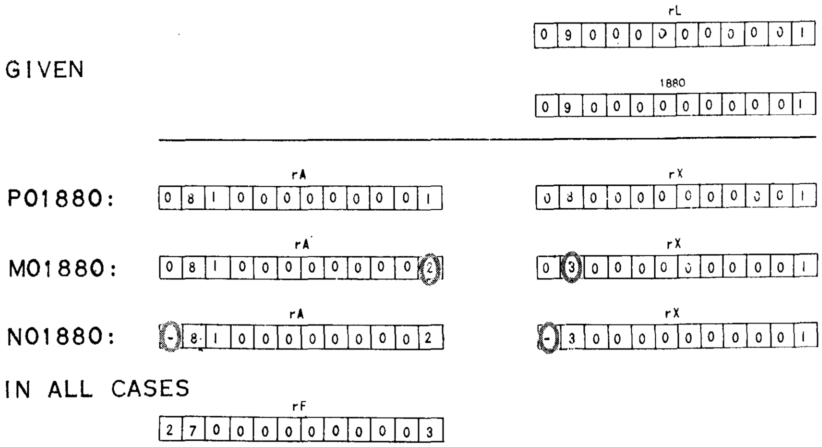


FIGURE 3-11

INSTRUCTION	OPERATION	MNEMONIC
DOm	(m)⟶rA; (rA) ÷ (rL)⟶rA [rounded], rX [unrouded]	Divide

Divide (m) by (rL). Transfer the rounded quotient to rA and the unrounded quotient to rX. [(rL) must be larger in absolute value than (m)]

The execution of the DOm instruction produces an 11 digit rounded quotient in rA and an 11 digit unrounded quotient in rX. The mnemonic is to <u>divide</u> (m) by (rL). The computer executes the DOm instruction as follows. (m) are transferred to rA. (rA) are divided by (rL). The unsigned, unrounded, 12 digit quotient is transferred to rX. Five is added to the least significant digit of (rX) and the sum is transferred to rA. (rA) and (rX) are shifted right one digit position. The sign of the quotient is transferred to the sign position of rA and rX.

For example:

```
                                    rL    030000000000
                                  0101    012345678917
        In executing D00101:

                411522630566              12 digit quotient without sign
                     ↙           \
            +000000000005          \
                  ↙                 \
          411522630571               ↘
                 ↙
       Shift 1 place right and insert sign
            ↙                       ↓
          rA                        rX

     0411522263057           041152263056
```

THE DECIMAL POINT

The computer fixes the decimal point between the sign and most significant digit positions. Because every algebraic number begins with a sign followed by a decimal point, as far as the computer is concerned, every algebraic quantity lies between plus one and minus one, the largest being

+.99999999999

the smallest

−.99999999999

How can algebraic quantities of magnitude one or larger, or minus one or less, be represented? This problem is really no different in kind than the similar one presented by an ordinary desk calculator. Like the computer, the calculator fixes the decimal point at some specific place, usually immediately after the least significant digit position. Yet operators have no difficulty in treating fractional quantities on a calculator. Such quantities are handled as follows. All quantities are entered into the calculator as whole numbers, and decimal points are assumed in the numbers to create the fractional quantities. During the calculation the assumed decimal points are ignored. After the calculation is complete, the decimal point is assumed in the result according to certain rules. The same kind of solution applies to the computer. Decimal points can be assumed in a word wherever wanted. At the end of the calculation the following rules apply.

RULE FOR ADDITION AND SUBTRACTION

To add two words, or to subtract one word from another, the decimal point must be assumed in the same place in both words. The word that represents the sum will have the assumed decimal point in the same place as it is assumed in the words entering the calculation.

A carat indicates the assumed decimal point.

```
$3600.05            036000500000          000000360005
                         ^                          ^
  156.23            001562300000          000000015623
                         ^                          ^
--------            ------------          ------------
$3756.28            037562800000          000000375628
                         ^                          ^
```

RULE FOR MULTIPLICATION

When multiplying one word by another, if the assumed decimal point is m digit positions to the right of the fixed decimal point in one word, and n position to the right in the other, the product will have the assumed point m plus n positions to the right.

RULE FOR DIVISION

When dividing one word by another, if the assumed point is m positions to the right of the fixed point in the dividend, and n positions to the right in the divisor, the quotient will have the assumed point m minus n positions to the right.

For example, if

$A = 0XXXX_{\wedge}XXXXXXX$ $\quad m = 4$

and $B = 0XXX_{\wedge}XXXXXXXX$ $\quad n = 3$

then

$AB = 0XXXXXXX_{\wedge}XXXX$ $\quad m + n = 7$

and $A \div B = 0X_{\wedge}XXXXXXXXXX$ $\quad m - n = 1$

If the assumed point is p positions to the left of the fixed point, it is - p positions to the right. The fact that assuming the decimal point p places to the right of the fixed point is equivalent to multiplying the word by 10^p makes the proof of the above rules immediate.

For example

$0_{\wedge}31200000000$ = .312 (no assumption made)

$031_{\wedge}200000000$ = $.312 \times 10^2 = 31.2$ (where the assumption is $p = 2$)

When n and/or m are zero the above rules give the following results. If m and n are zero then m plus n and m minus n are zero. Thus, if in two words, the decimal point is assumed at the fixed decimal point, the assumed decimal point in the product or quotient of the words will be at the fixed point.

If n is zero, then m plus n and m minus n equal m. Thus, if the point is assumed m positions to the right of the fixed point in a given word, and is assummed at the the fixed point in a second given word; the product of the given words, and the quotient of the first word divided by the second, will have the assumed decimal point m positions to the right. For example, if

	A = 0XXXXXXXXX‸XX	m = 9
and	B = 0‸XXXXXXXXXXX	n = 0
then		
	AB = 0XXXXXXXXX‸XX	m + n = 9
and	A ÷ B = 0XXXXXXXXX‸XX	m − n = 9

STUDENT EXERCISES

1. If A has the form 0‸XXXXXXXXXXX; and B, the form 0‸XXXXXXXXXXX; what is the form of AB and A ÷ B?

2. If A has the form 0XXXXXXX‸XXXX; and B, 0XXX‸XXXXXXXX; what is the form of AB and A ÷ B?

3. If A has the form 0XXXXXXXXX‸XX; and B, ‸0XXXXXXXXXXX; what is the form of AB and A ÷ B?

4. Reading the data stores three quantities of form

   0‸QQQQQQQQQQQ

   in cells 1880 - 1882. Print the product of the quantities.

5. Reading the data stores

DATA	FORM	CELL
Quantity A	0‸00AAAAAAAA	1880
Quantity B	0‸00BBBBBBBB	1881
Quantity C	0‸00CCCCCCCC	1882
Quantity D	0‸00DDDDDDDD	1883

If

$$E = AB$$

$$F = \frac{AB}{.9C}$$

$$G = \frac{AB}{.9C} - D$$

print E, F and G.

6. Reading the data stores

DATA	FORM	CELL
Income	0IIIIII‸II000	1880
Number of Dependents	00NN‸00000000	1881
Deductions other than for Dependents	000AAAA‸AA000	1882

A deduction of $600 is allowed for each dependent. The tax is twenty percent of taxable income. Print the tax in form

000000TTTT‸TT

THE CONTROL UNIT

The function of the control unit is to select instructions from the memory and execute them in proper sequence. The control unit is made up of three registers.

1. The Static Register (SR), a half word register.
2. The Control Register (CR), a one word register.
3. The Control Counter (CC), a one word register.

To execute an instruction the computer must transfer the instruction to the Static Register, the only place in the computer where an instruction can be interpreted. Since the computer can only execute one instruction at a time, only one instruction canbe stored in SR at any one time. Thus, SR is builtwith a six character capacity.

The computer transfers instructions from the memory to the control unit one word at a time and uses the Control Register to store the instruction pair while the instructions are waiting to be executed.

Having transferred an instruction pair from a given cell to CR, the computer must store the address of the cell immediately following the given cell in order that, when the instruction pair in CR has been executed, it will know in what cell to find the next pair. The computer stores this address in the four least significant digits of the word in the Control Counter.

In short,

1. SR is an interpretive device,
2. CR contains the current instruction pair
3. CC contains the address of the next instruction pair.

THE THREE STAGE CYCLE OF OPERATION

The computer's three stage cycle is as follows.

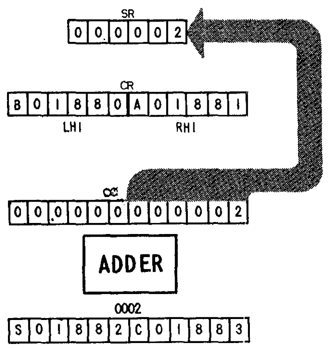
1. Transfer the next instruction pair to CR.
2. Execute the LHI.
3. Execute the RHI.

To transfer the next instruction pair to CR, the computer must know the address of this pair. This address is stored in the four least significant digits of (CC). To interpret this address, the computer must first transfer it to SR. Since SR has a six character capacity, the computer transfers the six least significant digits of (CC) to SR. The computer can then transfer to CR the contents of the cell specified by the four least significant digits of (SR).

At the same time that the computer transfers the current instruction pair to CR, it adds one to (CC) and transfers the sum to CC.CC then contains the address of the next instruction pair. To perform this addition the computer uses the algebraic adder, which only adds whole words. This fact is why CC has a one word capacity.

The stages of the cycle are called "beta (β) time","gamma (γ) time" and "delta (δ) time". For each stage to be executed, the proper information must first be transferred to SR. The time for this transfer is called "Time Out" (TO).

β TO — Transfer the six least significant digit of (CC) to SR.



β TIME OUT: (CC) ——→ SR

FIGURE 3-12

β Time On — Transfer to CR the contents of the cell specified by the four least significant digits of (SR). Add one to (CC), and transfer the sum to CC.

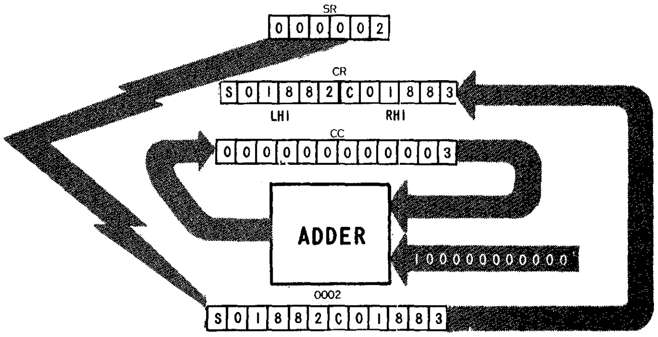


FIGURE 3-13 β TIME ON : (M) ——→ CR
(CC)+1 ——→ CC

γ TO — Transfer the LHI in CR to SR.

γ Time On — Execute the instruction in SR.

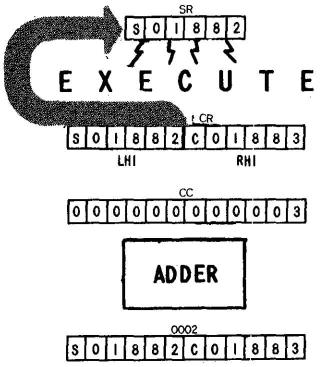
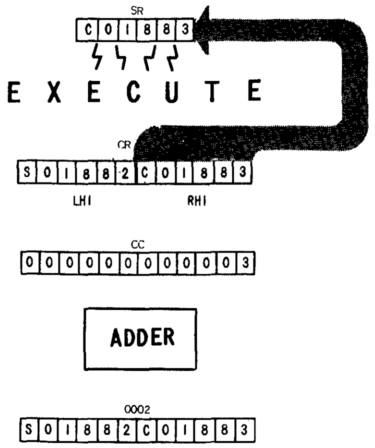


FIGURE 3-14 γ TIME OUT : LHI ——→ SR
TIME ON : EXECUTE

δ TO Transfer the RHI in CR to SR.

δ Time On Execute the instruction in SR.



δ TIME OUT : RHI ——▶ SR
TIME ON : EXECUTE

FIGURE 3-15

The cycle is then repeated for the next instruction pair.

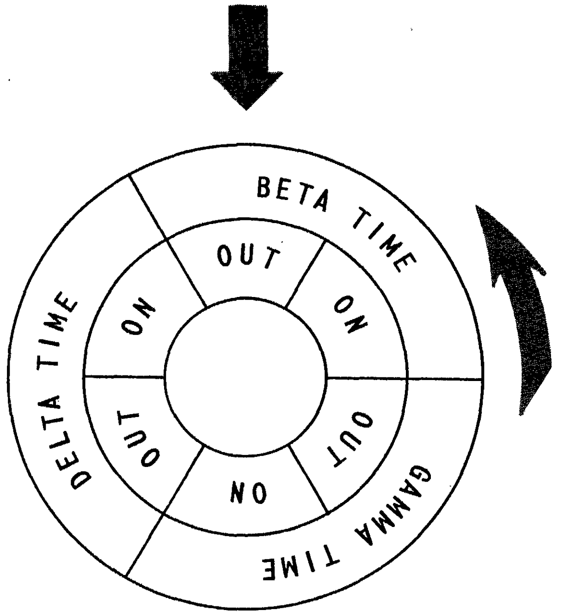


FIGURE 3-16

The elements of the memory, control and arithmetic units and their interrelations are shown in the following figure.

COMPLETION OF FIRST CYCLE

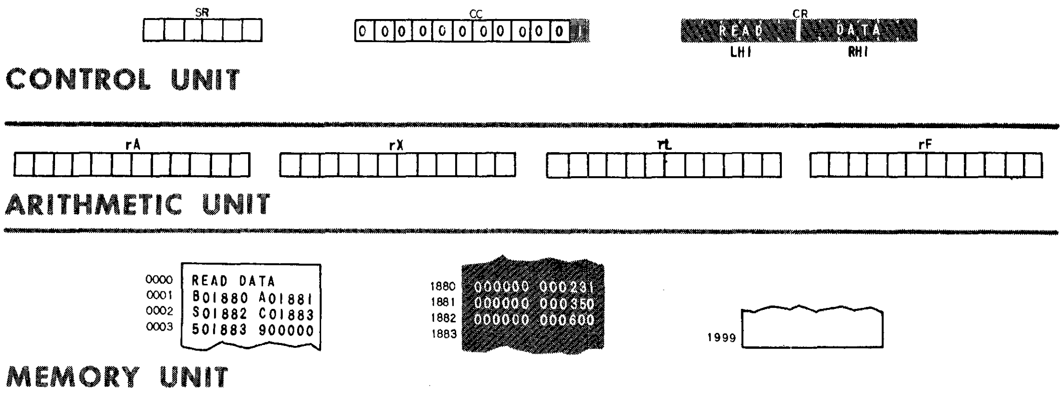


FIGURE 3-20

On β TO the six least significant digits of (CC), 000001, are transferred to SR.

SECOND CYCLE: COMPLETION OF β TO

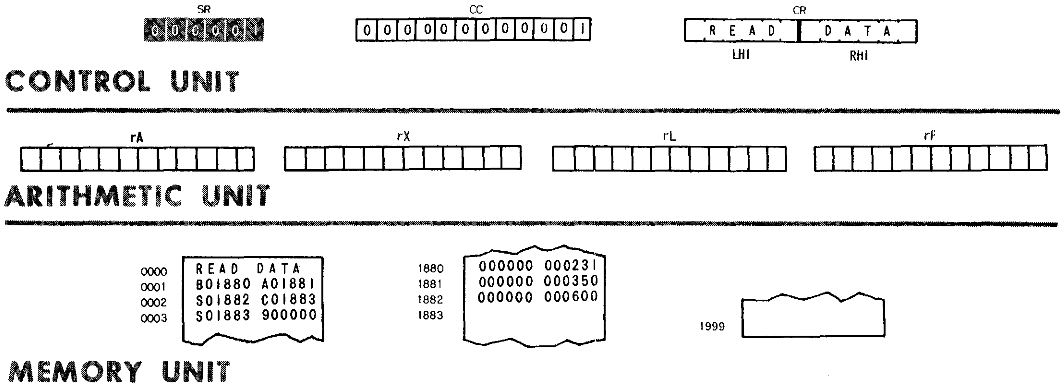


FIGURE 3-21

On β Time On the contents of the cell specified by the four least significant digits of (SR), cell 0001, are transferred to CR, and one is added to (CC), the sum being transferred to CC.

SECOND CYCLE: COMPLETION OF β TIME ON

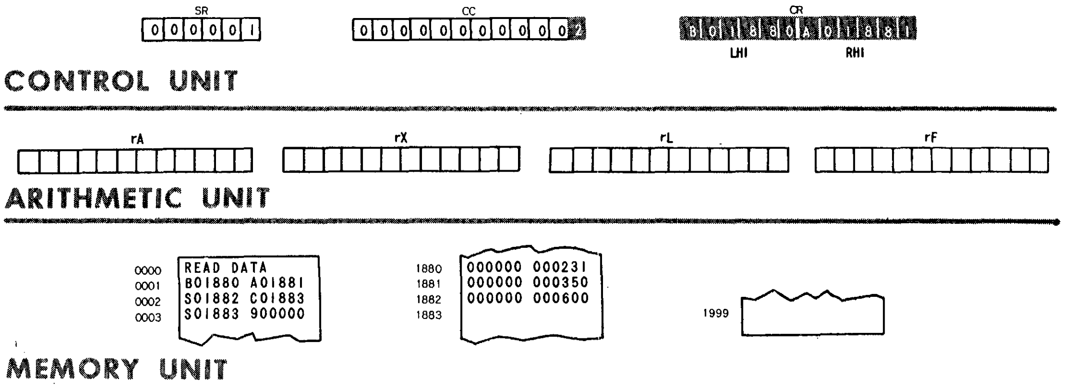


FIGURE 3-22

On γ TO the LHI in CR, B01880, is transferred to SR.

SECOND CYCLE: COMPLETION OF Y TO

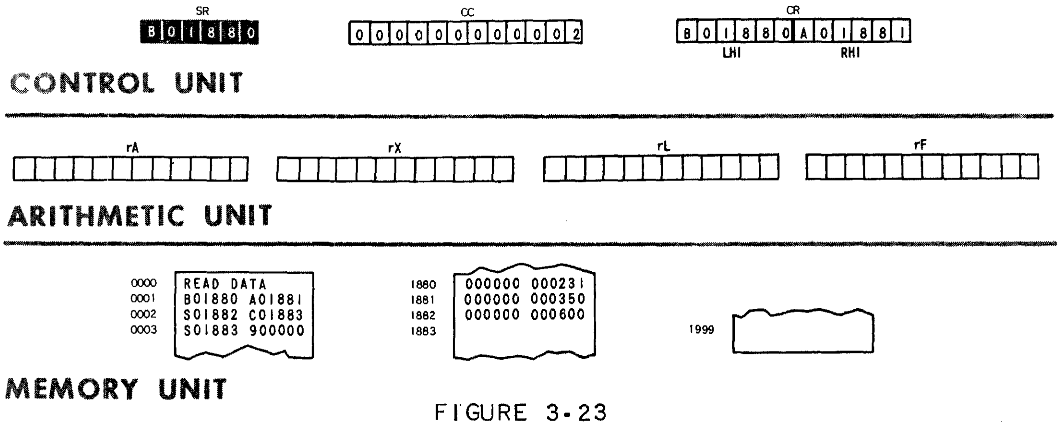


FIGURE 3-23

On γ Time On the instruction in SR,B01880, is executed.

SECOND CYCLE: COMPLETION OF γ TIME ON

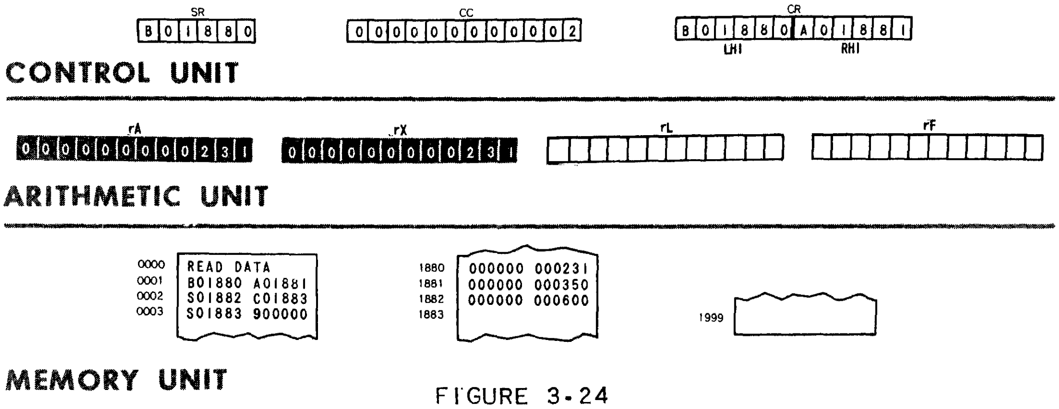


FIGURE 3-24

On δ TO the RHI, A01881, is transferred to SR.

SECOND CYCLE: COMPLETION OF δ TO

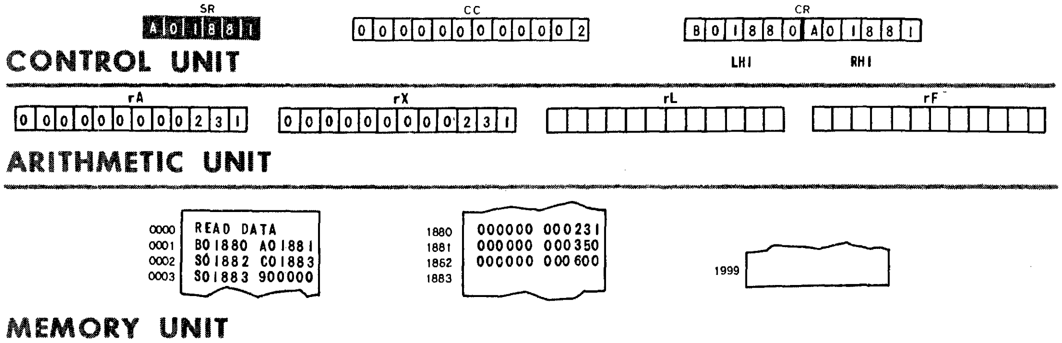


FIGURE 3-25

On δ Time On the instruction, A01881, is executed.

SECOND CYCLE: COMPLETION OF δ TIME ON

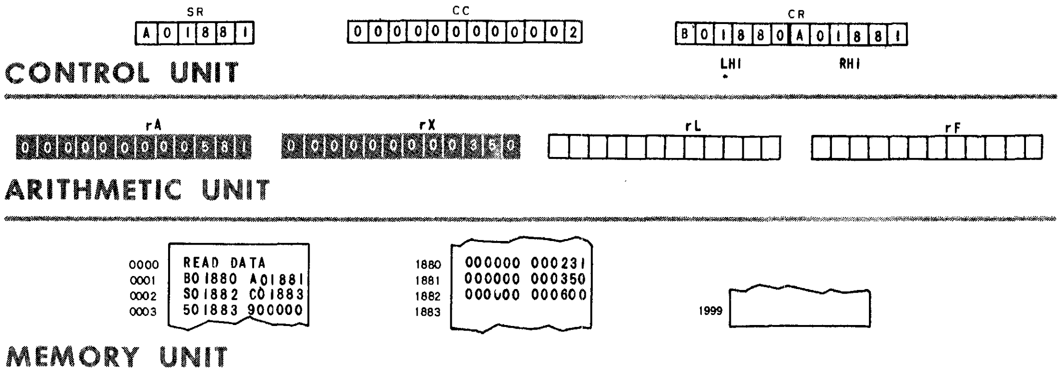


FIGURE 3-26

On β TO the contents of CC, 000002, are transferred to SR.

THIRD CYCLE: COMPLETION OF β TO

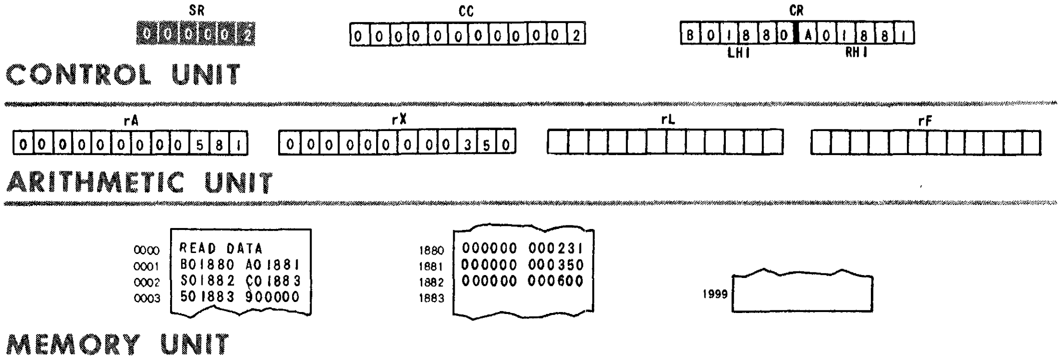


FIGURE 3-27

On β Time On the contents of the cell specified by the contents of SR, cell 0002, are transferred to CR and one is added to (CC).

THIRD CYCLE: COMPLETION OF β TIME ON

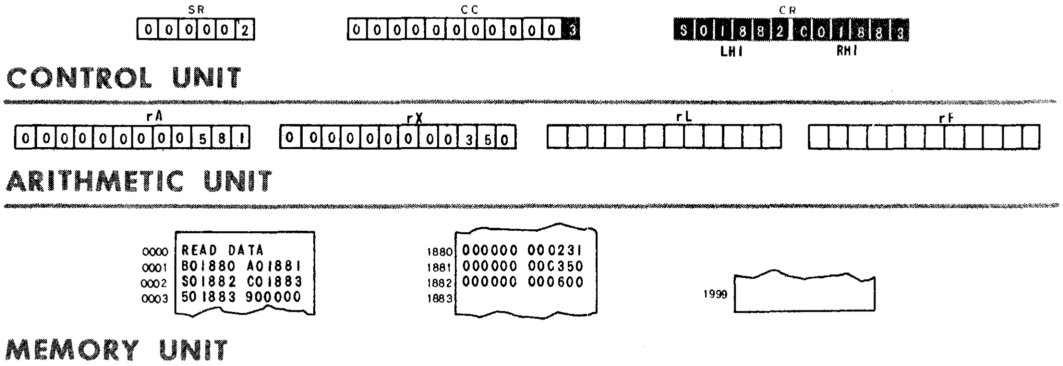


FIGURE 3-28

On γ Time the instruction S01882 is transferred to SR and executed.

THIRD CYCLE: COMPLETION OF γ TIME

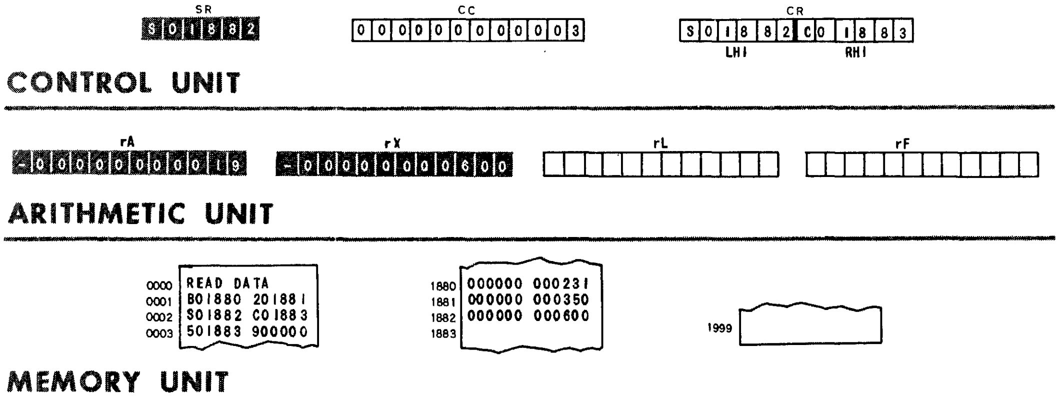


FIGURE 3-29

On δ Time C01883 is transferred to SR and executed.

THIRD CYCLE: COMPLETION OF δ TIME

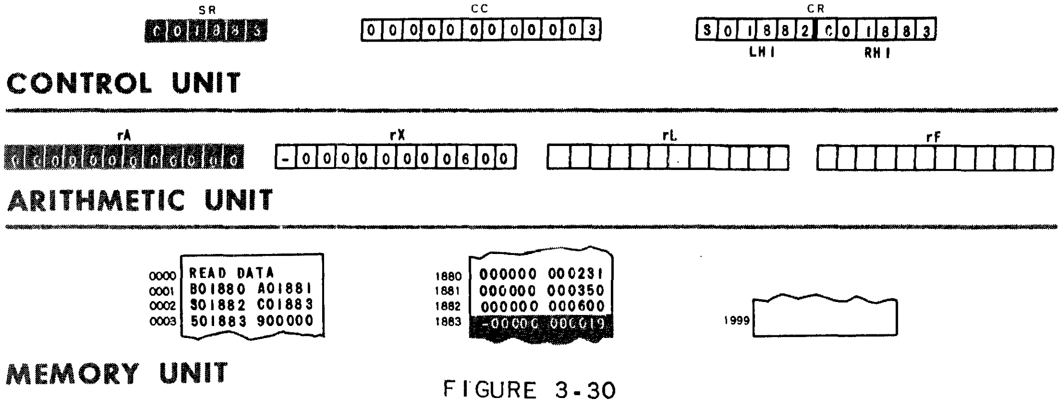


FIGURE 3-30

On β Time 000003 is transferred to SR; the contents of the cell specified, cell 0003, are transferred to CR; and (CC) are increased by one.

FOURTH CYCLE: COMPLETION OF β TIME

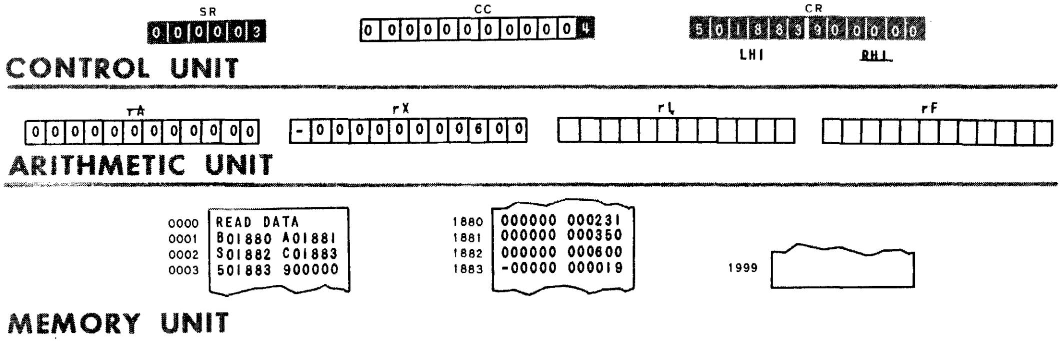


FIGURE 3-31

On γ Time 501883 is executed, printing -00000000019.

FOURTH CYCLE: COMPLETION OF γ TIME

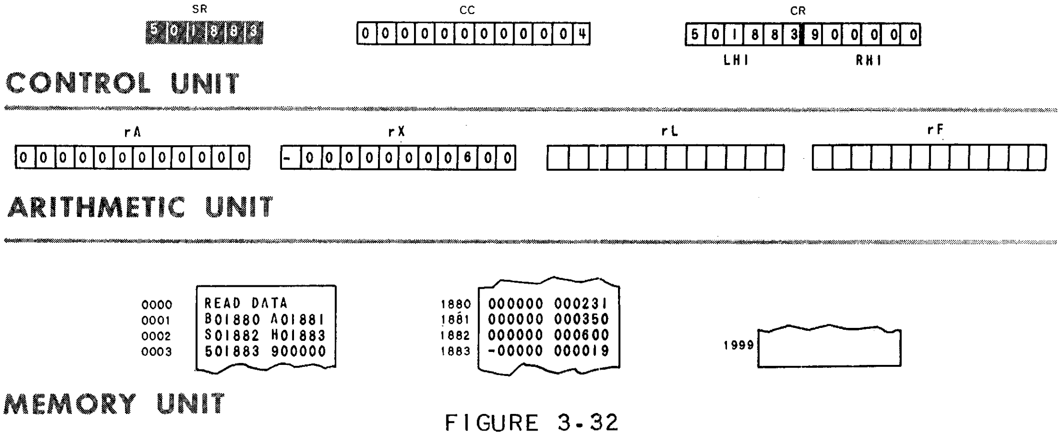


FIGURE 3-32

On δ Time 900000 is transferred to SR and executed, stopping the computer.

FOURTH CYCLE: COMPLETION OF δ TIME

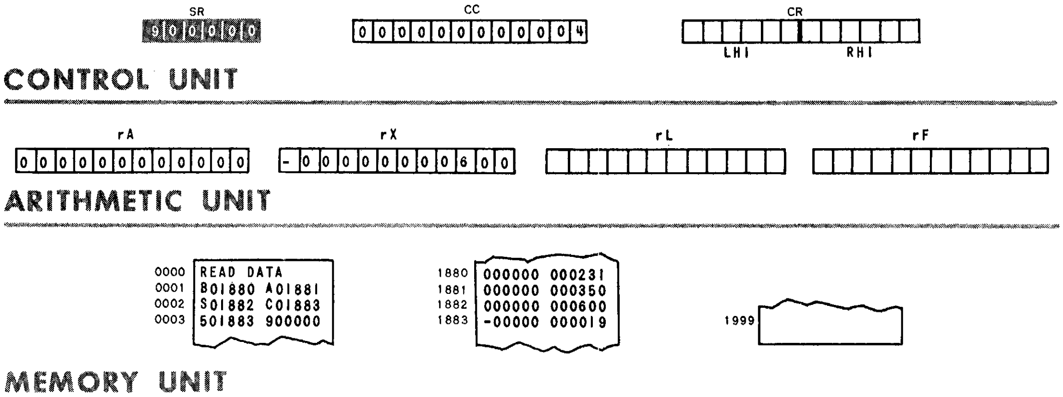


FIGURE 3-33

TRANSFER OF CONTROL INSTRUCTIONS

Having executed the instruction pair in cell "k", it is sometimes advantageous for the next instruction pair to be in a cell other than cell "k+1". This breaking of the computer's sequential operation is called transfer of control.

INSTRUCTION	OPERATION	MNEMONIC
UOm	00000000 (CR) ⟶ CC	Unconditional Transfer of Control

Transfer control to m. Sequential operation is broken at cell k and resumes at cell m.

The mnemonic is unconditional transfer of control to m, since the execution of the UOm instruction results in transfer of control regardless of the conditions present in the computer. The computer executes the UOm instruction as follows. CC contains the address of the next instruction pair. If the execution of the UOm instruction is to transfer control to m, the execution must transfer the address part of the UOm instruction to CC. Actually, the UOm instruction is executed by transferring the four least significant digits of (CR) to the four least significant digit positions of CC. This method of execution will achieve the purpose of the UOm instruction provided that the address part of the UOm instruction is the four least significant digits of the word in which the UOm instruction appears. In effect, this fact means that the UOm instruction should be coded as a RHI.

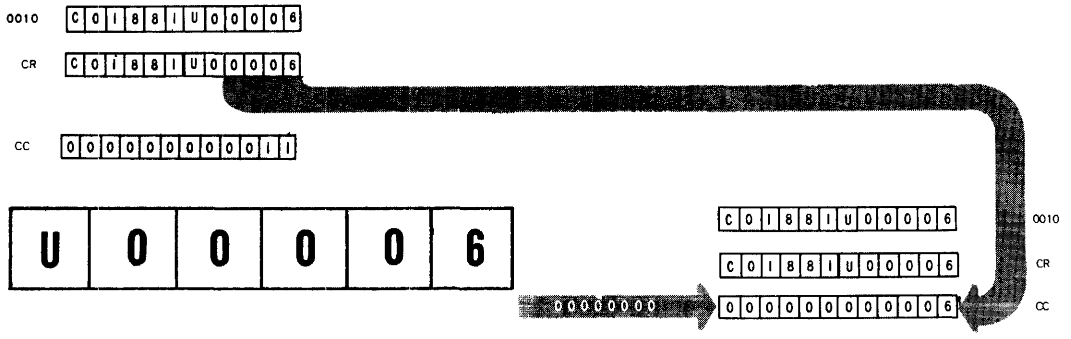


FIGURE 3-34

If a UOm instruction is properly coded in cell k, when the instruction pair in cell k has been executed, the next pair of instructions to be executed are not in cell k+1, but in cell m.

Consider the following.

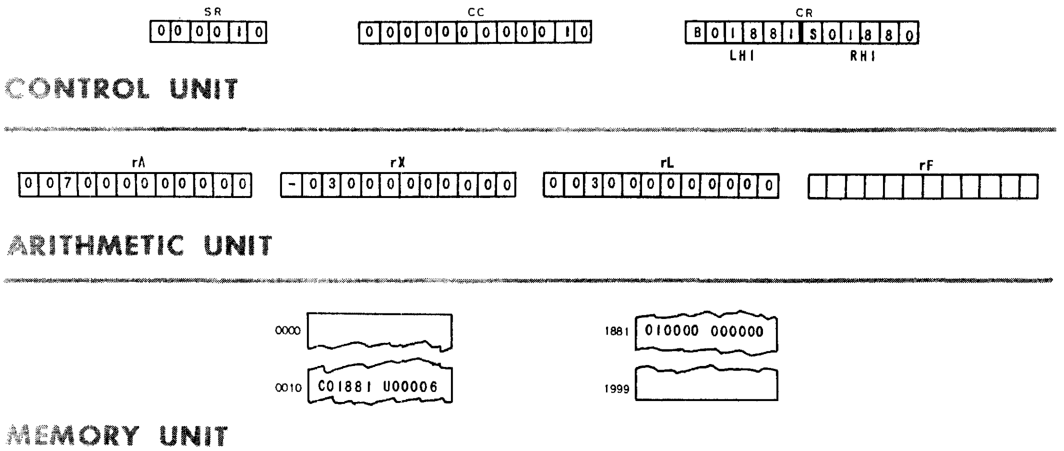


FIGURE 3-35

Assume that the computer has just completed β TO. On β Time On the contents of cell 0010, C01881U00006, are transferred to CR, and (CC) are increased by one.

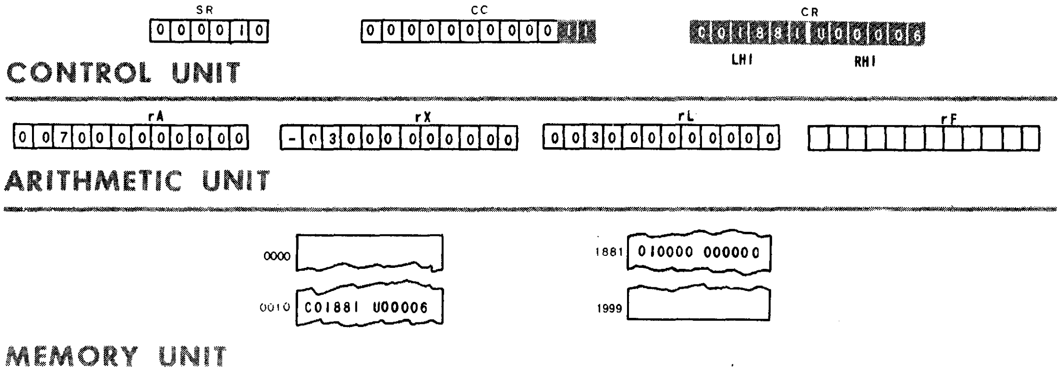


FIGURE 3-36

CR contains the current instruction pair, and the four least significant digits of (CC) specify that the next instruction pair is in cell 0011. On γ Time On C01881 is executed.

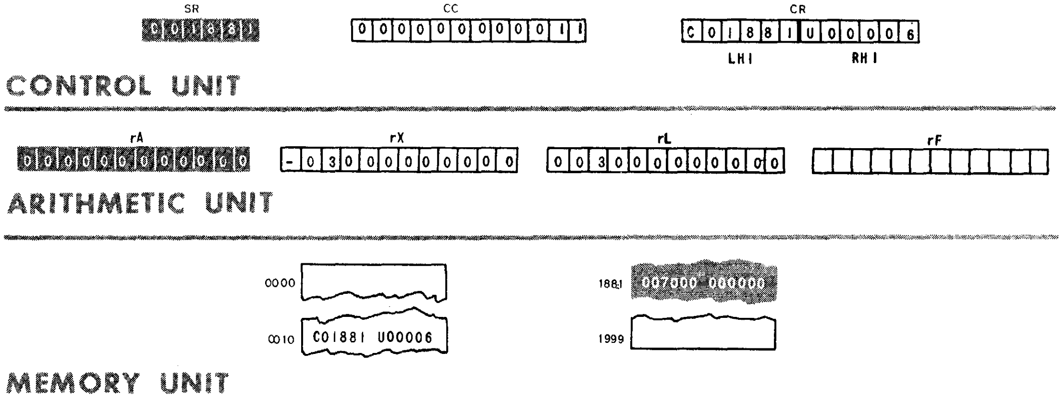


FIGURE 3-37

On δ Time, U00006.

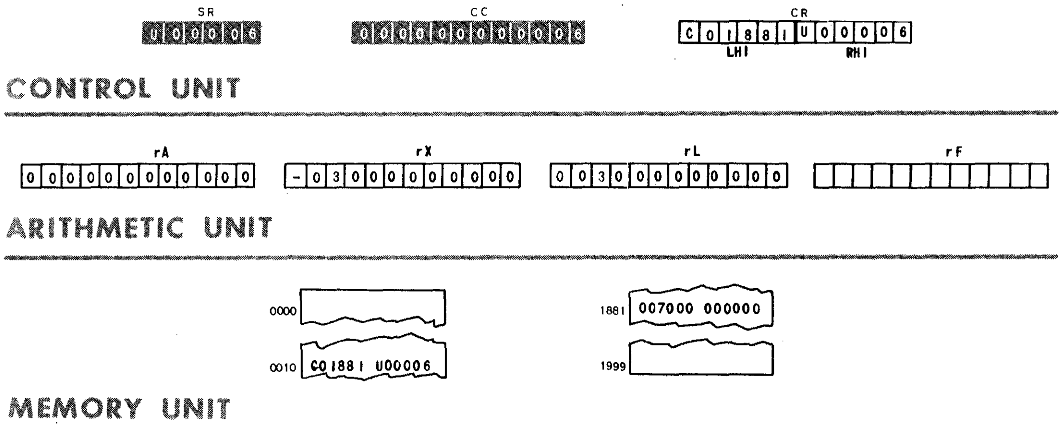


FIGURE 3-38

The four least significant digits of (CC) no longer specify that the next instruction pair is in cell 0011, but instead specify that the pair is in cell 0006. The computer's sequential operation has been broken, and control has been transferred to cell 0006. With the instruction pair in cell 0006 the sequential operation will resume and continue until another transfer of control or stop instruction is executed.

INSTRUCTION	OPERATION
OOm	Skip

Pass control to the next stage of the three stage cycle.

In executing the OOm instruction, the computer ignores m. The execution of the OOm instruction does not alter the contents of any cell or register. One use of the OOm instruction is as follows. The situation may arise where the next instruction to be coded is both a LHI and a UOm instruction. To be coded properly, the UOm instruction should be coded as a RHI. Yet the computer cannot skip a stage of its three stage cycle and must have some instruction to execute on δ Time. The OOm instruction is used in such situations.

In contrast to the UOm instruction are the conditional transfer of control instructions.

INSTRUCTION	OPERATION	MNEMONIC
QOm	If (rA) = (rL), then QOm acts as UOm; if not,as OOm	Equality Transfer of Control

If (rA) are identical to (rL), interpret QOm as UOm; if not, as OOm.

The mnemonic is: on equality of (rA) and (rL), control is transferred.

INSTRUCTION	OPERATION	MNEMONIC
TOm	If (rA) > (rL), then TOm acts as UOm; if not, as OOm	Threshold Transfer of Control

If (rA) are greater than (rL),
interpret TOm as UOm; if not, as OOm.

The mnemonic is: if (rA) are greater than the threshold set up by (rL), control is transferred.

(rA)	(rL)	Does the TOm Instruction Transfer Control?
012 345 678 910	009 761 835 011	Yes
-12 345 678 910	009 761 835 011	No
012 345 678 910	-99 999 999 999	Yes
-12 345 678 910	-99 999 999 999	Yes

For purposes of the TOm instruction an order of magnitude has been assigned to all characters. In figure 2-18, reading down the first column, then down the second, then the third, and finally the fourth, is equivalent to reading the characters in their ascending order of magnitude. The smallest character is i, the largest is = .

(rA)	(rL)	Does the TOm Instruction Transfer Control?
0BCDEFGHIJKL	023456789ABC	Yes
- BCDEFGHIJKL	023456789ABC	No
0BCDEFGHIJKL	-DEFGHIJKLMN	Yes
- BCDEFGHIJKL	-DEFGHIJKLMN	Yes

If (rA) and (rL) have signs, the TOm instruction treats both quantities as signed numbers. If either word has no sign, the TOm instruction treats the words in their entirety.

(rA)	(rL)	Does the TOm Instruction Transfer Control?
0123456789AB	234567890ABC	No
34567890ABCD	-567890ABCDE	Yes
67890ABCDEFG	7890ABCDEFGH	No

The function of the conditional transfer of control instructions is to allow the computer to choose between different processing possibilities dependent on the nature of the data.

Illustrative Example

Reading the data stores

DATA	FORM	CELL
Account Number	0AAAAAAAAAAA	1880
Delinquent Account Number	0DDDDDDDDDDD	1881

If the account number is equal to the delinquent account number print

ΔNOΔCREDITΔ

If not, print

CREDITΔGOOD.

LOGICAL ANALYSIS

1. Read the data
2. Is the account number equal to the delinquent account number?

2a.No	2b. Yes
3. Print CREDIT GOOD	3. Print NO CREDIT

4. Stop

CODING

```
0000    READ              }     Read the data
                  DATA    }
0001    B01880            |
                  L01881  |     Is the account number equal to
0002    ‿                 |     the delinquent account number?
                  Q00004  |
______________________
0003    500005                  Print CREDIT GOOD.
                  900000        Stop
______________________
0004    500006                  Print NO CREDIT
                  900000        Stop
______________________
0005    CREDIT            |
                  GOOD.   |     Constants
0006    ΔNOΔCR            |
                  EDIT.Δ  |
```

For ease in writing, a LHI or RHI consisting of six zeros in customarily written as ‿. It is also customary to draw a line under all transfer of control instructions.

The following is a description of the thinking that might have accompanied this coding. The student exercises begin on page 69 .

After the read data and the execution of the BOm and LOm instructions, the proper quantities are in rA and rL, and the QOm instruction can be coded. But the next instruction to be coded is a LHI. Since the QOm instruction can be interpreted as a UOm instruction, to be properly coded, the address part of the QOm instruction must be the four least significant digits of the word in which the QOm instruction appears. The simplest way to achieve this situation is to code a OOm instruction for the LHI.

It makes no difference what cell is specified by the QOm instruction as long as the processing called for by the condition of equality begins in that cell. To conserve memory space it is convenient not to specify any cell at this time, and instead, code the processing called for by the condition of inequality, which must begin in cell 0003.

The execution of a 50m instruction is required to print CREDIT GOOD. It makes no difference what cell is specified by the 50m instruction as long as the word

CREDIT∆GOOD.

is stored in it. It is convenient not to specify any cell at this time, and instead continue the coding. The 90m instruction completes this logical branch of the coding.

The next free cell is cell 0004, which can be specified by the QOm instruction. A 50m instruction and a 90m instruction in cell 0004 complete the coding.

The next free cells are cells 0005 and 0006, which can be specified by the 50m instructions.

STUDENT EXERCISES

1. Reading the data stores:

DATA	FORM	CELL
Pay	000000PPPP‸PP	1880
Deduction	00000000DD‸DD	1881

If the deduction does not reduce the pay to less than $15, make the deduction; other wise, print the deduction. In either case, print the pay.

2. Reading the data stores a charge in the form:

000000CCCC‸CC

If the charge is greater than or equal to $150.00, apply a discount of three percent, and print the resulting charge. Otherwise, print the original charge.

3. Reading the data stores

DATA	FORM	CELL
Stock Number	NNNNNNNNNNNN	1880
On Hand	000000⊙⊙⊙⊙‸⊙⊙	1881
Sold	000000SSSS‸SS	1882
Minimum Required	000000RRRR‸RR	1883

Update the on hand. If the sales reduce the on hand below the required, print the stock number.

4. Reading the data stores

DATA	FORM	CE LL
Quantity Ordered	0000QQQQ‸QQ00	1880
Unit Price	0PP‸PP0000000	1881

If the quantity is greater than or equal to 100, apply a discount of 40%. Otherwise, apply a discount of 30%. Print the charge.

SUMMARY

Instruction format: Of the six digits the first two indicate the operation to be performed on the word in the cell whose address is that of the last four instruction digits.

INSTRUCTION	OPERATION	MNEMONIC
AOm:	$(m) \rightarrow rX;\ (rA) + (rX) \rightarrow rA$	Add
BOm:	$(m) \rightarrow rA;\ rX$	Bring
COm:	$(rA) \rightarrow m;\ 0 \rightarrow (rA)$	Clear
DOm:	$(m) \rightarrow rA;\ (rA) \div (rL) \rightarrow rA$ rounded	Divide
	$\rightarrow rX$ unrounded	
HOm:	$(rA) \rightarrow m$	Hold
I Om:	$(rL) \rightarrow m$	Into
JOm:	$(rX) \rightarrow m$	—

INSTRUCTION	OPERATION	MNEMONIC
KOm:	(rA)→rL; 0→rA, ignore m	—
LOm:	(m)→rL, rX	Load
MOm:	(m)→rX; (rL) x (rX)→rA, rounded	Multiply
NOm:	- (m)→rX; (rL) x (rX)→rA, rounded	Negative Multiply
POm:	(m)→rX; (rL) x (rX)→rA, rX, 22 digits	Precision Multiply
QOm:	if (rA) = (rL), transfer control to m	Equality Transfer
S Om:	− (m)→rX, (rA) + (rX)→rA	Subtract
TOm:	if (rA) > (rL), transfer control to m	Threshold Transfer
UOm:	transfer control to m	Unconditional Transfer
XOm:	(rA) + (rX)→< A	—
OOm:	skip, ignore m	—
5 0 m:	print (m) on the SCP	—
9 0 m:	stop, ignore m	—

CONTROL UNIT

Control Counter - 1 word register - holds the address of the next pair of instructions.
Control Register - 1 word register - holds the current pair of instructions.
Static Register - one half word register - decodes or interprets one instruction or address.

Three Stage Cycle of Operation

β - (CC)→SR
(m)→CR
(CC) + 1→CC

γ - LHIcr→SR and execute
δ - RHIcr→SR and execute

chapter 4

Introduction to Flow Charts

EXAMPLE

Reading the data stores:

DATA	FORM	CELL
Days of Medical Absence	0AA‸000000000	1880
Days of Allowable Medical Leave Remaining	0LL‸000000000	1881
Hourly Rate of Pay	0RRR‸RR000000	1882

Update the medical leave, and print the employee's medical pay in form

00000000PP‸PP

LOGICAL ANALYSIS

```
 1. Read data.
 2. Is medical absence equal to zero?
2a No.                                                        | 2b Yes.
 3. Is medical leave equal to zero?                           |
3a No.                                            | 3b Yes.   |
 4. Is medical leave greater than medical absence?|
4a No.                          | 4b Yes          |
 5. Store medical leave in      | 5. Store medical absence
    storage.                    |    in storage.  |
 6. Store zero in medical leave.| 6. Reduce medical leave
 7. Multiply storage by eight.  |    by medical absence.
 8. Multiply product by rate.                     |
 9. Print product.                                | 9. Print zero.
10. Stop.
```

This analysis is precise but bulky. As the size and .complexity of problems increase such written analyses would become less and less helpful because of the large amount of writing necessary.

The analysis can be made clearer by putting the steps in boxes and using arrows to indicate the sequence of steps.

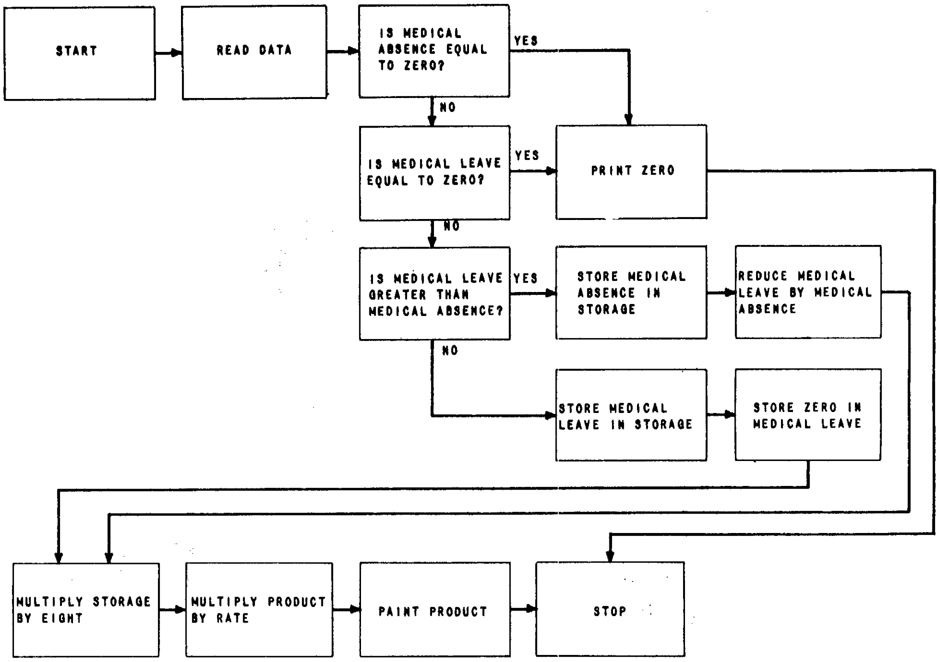

START
READ DATA
IS MEDICAL ABSENCE EQUAL TO ZERO?
YES
NO
IS MEDICAL LEAVE EQUAL TO ZERO?
YES
PRINT ZERO
NO
IS MEDICAL LEAVE GREATER THAN MEDICAL ABSENCE?
YES
STORE MEDICAL ABSENCE IN STORAGE
REDUCE MEDICAL LEAVE BY MEDICAL ABSENCE
NO
STORE MEDICAL LEAVE IN STORAGE
STORE ZERO IN MEDICAL LEAVE
MULTIPLY STORAGE BY EIGHT
MULTIPLY PRODUCT BY RATE
PAINT PRODUCT
STOP

FIGURE 4-1

This solution to the problem of picturing the analysis is superior but would still result in a massive chart for a large problem. A furthei reduction can be made by using letters to denote the quantities processed and arithmetic symbols to define the processing. The use of symbols requires a legend on the analysis to define each letter so there will be no confusion as to the nature of the quantities.

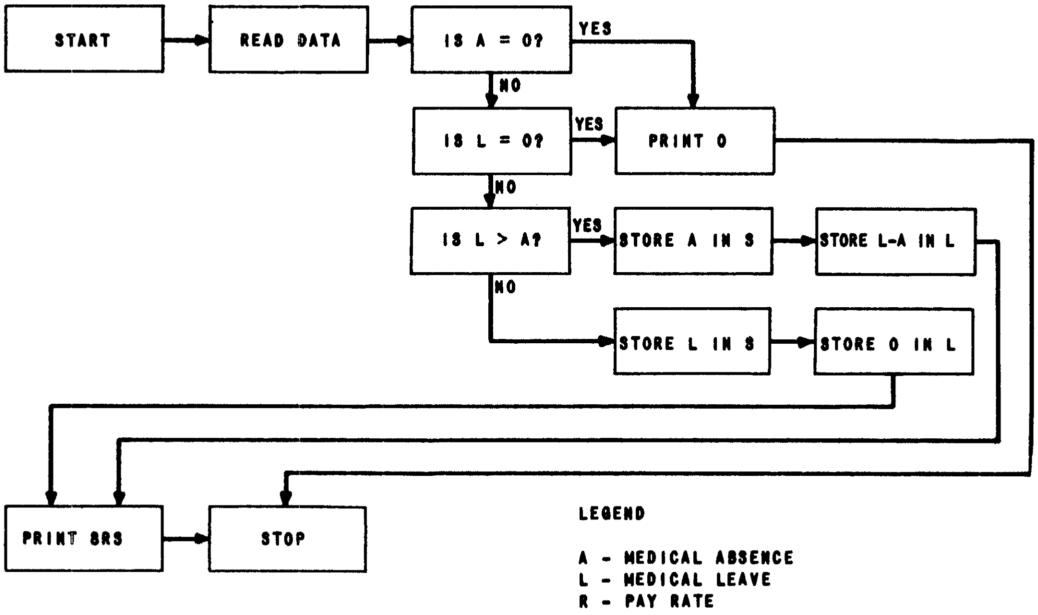


FIGURE 4-2

An analysis involves, at most, three types of processing.

1. Transfer of data.
2. Arithmetic operations.
3. Logical decisions.

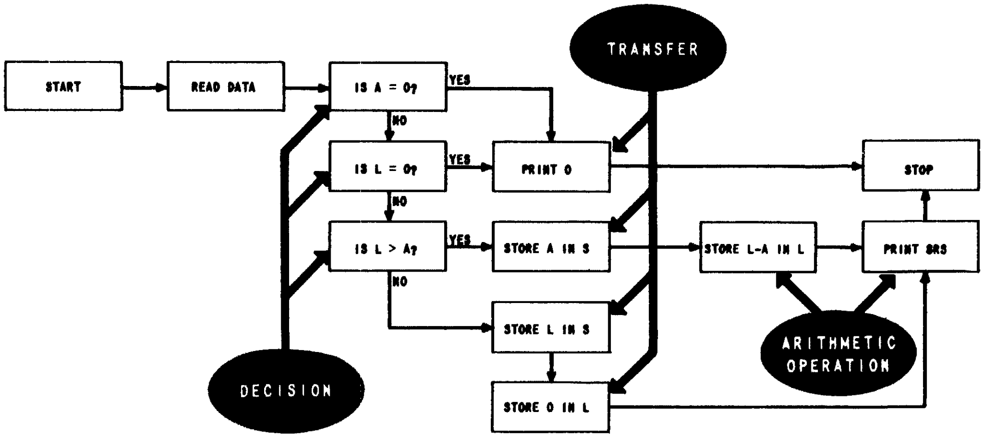


FIGURE 4-3

To further reduce the size of the analysis, transfers and arithmetic operations will be shown in rectangles. The distinguishing feature of a transfer or arithmetic operation is the inclusion of an arrow in the rectangle to indicate the substitution of one quantity for another.

Decisions are shown in flattened ovals. The distinguishing features of a decision are:

1. The inclusion of a colon in the oval to indicate the comparison of one quantity with another

and 2. Two arrows coming out of the oval to indicate that, on the basis of the decision, one of two possible paths of processing will be followed.

Each of these paths is labelled with the condition which must exist for that path to be followed.

The decision "is A = 0" (yes or no) will be shown as

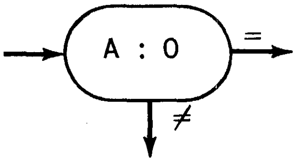


FIGURE 4-4

If the two quantities are equal, the next step follows the arrow labelled with the equal sign; if unequal, it follows the arrow labelled with the unequal sign.

The decision "is L > A" (yes or no) will be shown as

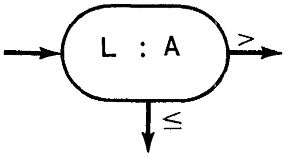


FIGURE 4-5

If L is greater than A, the next step follows the arrow labelled with the "greater than" sign (>); if L is not greater than A (i.e., less than or equal to A), the next step is written following the arrow labelled with the "less than or equal to" sign(≤).

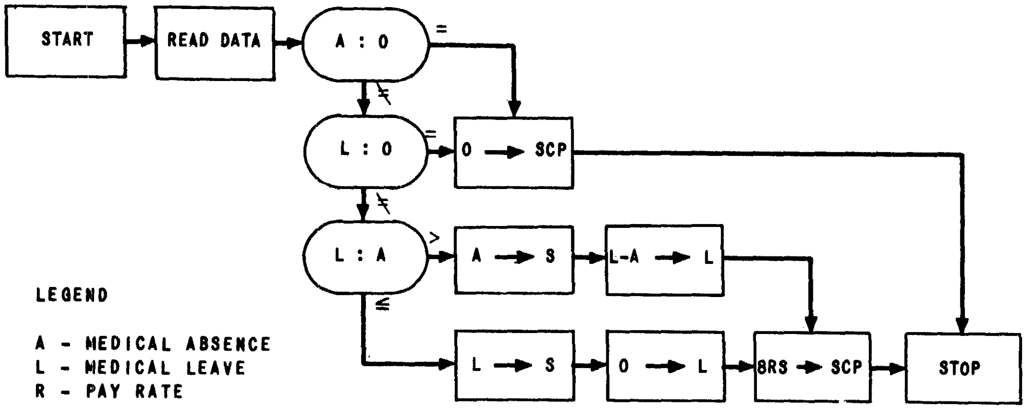


FIGURE 4.6

To reduce the length of the arrows indicating the sequence of steps, "fixed connectors" are used. A fixed connector is a numbered circle. When an arrow leads to a fixed connector,

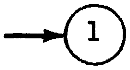


FIGURE 4.7

the next step follows the arrow leading out of the fixed connector enclosing the same number.

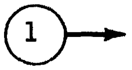


FIGURE 4.8

Thus,

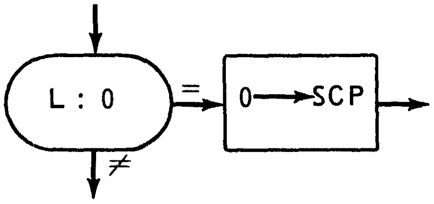


FIGURE 4.9

is the same as

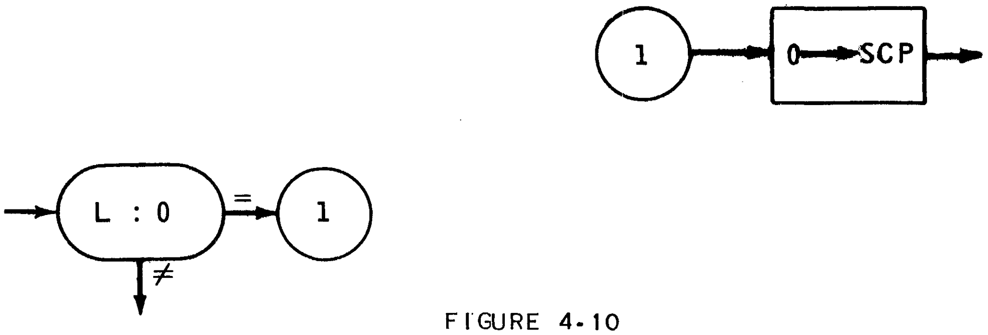


FIGURE 4-10

A fixed connector enclosing a given number and having an arrow leading to it can appear in an analysis as many times as is necessary, but to avoid ambiguity, only one fixed connector enclosing the number and having an arrow leading out of it can appear.

The boxes enclosing the words, start and stop, serve a function similar to that of connectors in that they show the beginning and the end of the processing steps. The start and stop are also shown in circles.

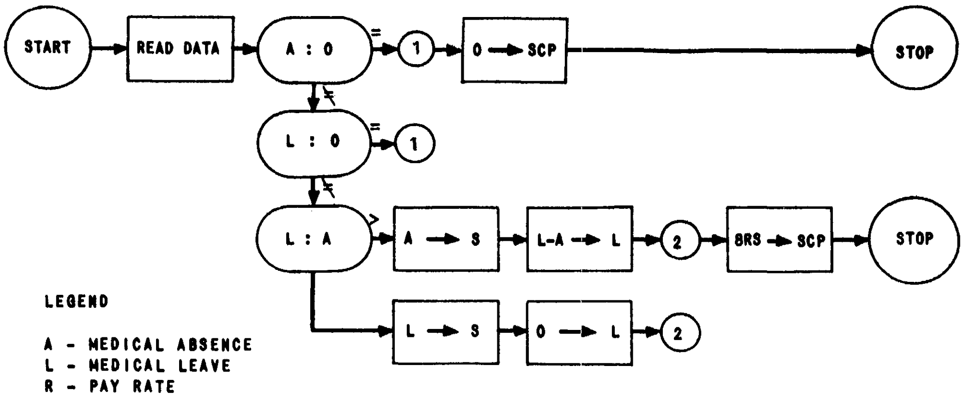


FIGURE 4-11

Often, parts of a complex analysis extend beyond the paper on which the analysis is made. When such a situation arises, a fixed connector can be used to direct the processing to a point on the paper where there is room – a mechanism analogous to the carriage return on a typewriter.

An analysis drawn according to the above conventions is called a "flow chart."

CODING

```
     0000   READ                   read data
                       DATA
     0001   B01880              ⎫
                       L00012   ⎬  A : 0
     0002   ~                   ⎪
                       Q00011   ⎭
     ----------------------------
     0003   B01881              ⎫
                       Q00011   ⎭  L : 0
     ----------------------------
     0004   L01880              ⎫
                       T00009   ⎭  L : A
     ----------------------------
     0005   K00000                 L→S
                       C01881      0→L
  ②  0006   M00013              ⎫
                       K00000   ⎪
     0007   M01882              ⎬  8RS→SCP
                       C01883   ⎪
     0008   501883              ⎭
                       900000      Stop
     ----------------------------
     0009   S01880              ⎫
                       C01881   ⎭  L − A→L
     0010   ~
                       U00006
     ----------------------------
     0011   500012                 0→SCP
                       900000      Stop
     ----------------------------
     0012   ~
                       ~
     0013   000080
                       ~
```

ILLUSTRATIVE EXAMPLE

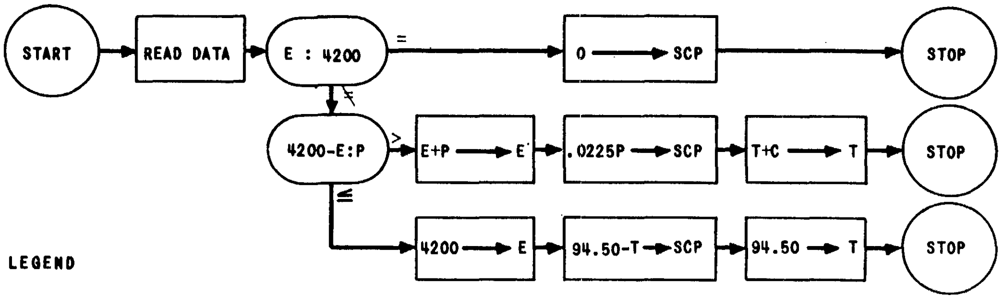
Reading the data stores

DATA	FORM	CELL
Year-to-Date FICA Earnings	000000EEEE‸EE	1880
Year-to-Date FICA Tax	00000000TT‸TT	1881
Current Pay	000000PPPP‸PP	1882

Update the year-to-date FICA earnings and tax, and print the current FICA tax in form

00000000CC‸CC

FLOW CHART



LEGEND

E - YEAR TO DATE FICA EARNINGS
T - YEAR TO DATE FICA TAX
P - CURRENT PAY

FIGURE 4-12

CODING

```
0000    READ                      read data
                  DATA
0001    B01880
                  L00015          E : 4200
0002    ~.
                  Q00014

0003    B00015
                  S01880          4200 - E : P
0004    L01882
                  T00010

0005    B00015                    4200 → E
                  C01880
0006    B00016
                  S01881          94.50 - T → SCP
0007    C01883
                  B00016
0008    501883                    94.50 → T
                  C01881
0009    900000                    Stop
                  ~
0010    B01880
                  A01882          E + P → E
0011    C01880
                  M00017          .0225 P → SCP
0012    H01883
                  A01881          T + C → T
0013    ~
                  U00008
```

```
0014    500018                 0——►SCP
                   900000      Stop
-----------------------------
0015    ⁓
                   420000
0016    ⁓
                   009450
0017    002000
                   ⁓
0018    ⁓
                   ⁓
```

STUDENT EXERCISES

Flow chart and code the following.

1. Reading the data stores a quantity of form

   ±QQQQQQQQQQ$_\wedge$

   in cell 1880. If the quantity is negative,print

   ΔΔNEGATIVE.Δ

   if positive, but less than 500,

   ΔΔΔSMALL.ΔΔΔ

   if greater than or equal to 500, but less than 1000,

   ΔΔΔMEDIUM.ΔΔΔ

   if greater than or equal to 1000,

   ΔΔΔLARGE.ΔΔΔ

2. Reading the data stores three quantities of form

   0QQQQQQQQQQ$_\wedge$

   in cells 1880 - 1882.

   Print the smallest of the quantities.

3. Reading the data stores

DATA	FORM	CELL
Badge Number	NNNNNNNNNNNN	1880
Bond Deduction	$\mathrm{00000000DD}_{\wedge}\mathrm{DD}$	1881
Cumulative Bond Deduction	$\mathrm{0000000CCC}_{\wedge}\mathrm{CC}$	1882
Bond Price	$\mathrm{0000000\ PPP}_{\wedge}\mathrm{PP}$	1883

Update the cumulative bond deduction, and if a bond can be purchased, print the badge number and the bond price.

4. Reading the data stores

DATA	FORM	CELL
Salesman's Number	NNNNNNNNNNNN	1880
Quantity Sold	$\mathrm{000000\ QQQQ}_{\wedge}\mathrm{QQ}$	1881
Unit Price	$\mathrm{0PP}_{\wedge}\mathrm{PP0000000}$	1882

If more than 50 units are sold, a discount of 10% is applied to the entire order. The salesman receives a 5% commission on the charge to the customer. Print the salesman's number and commission in form

$$\mathrm{0000000CCC}_{\wedge}\mathrm{CC}$$

5. Reading the data stores a employee's pay of form

$$\mathrm{000000PPPP}_{\wedge}\mathrm{PP}$$

in cell 1880. The percentage tax is given in the following table.

PAY	TAX PERCENTAGE
$ 1 - 14 99	1%
15 00 - 29 99	2%
30 00 - 44 99	3%
45 00 - 59 99	4%
60 00 or over	5%

Deduct the tax, and print the net pay in form

$$\mathrm{000000NNNN}_{\wedge}\mathrm{NN}$$

6. Reading the data stores

DATA	FORM	CELL
Year-to-Date Sales	0000SSSSSS‸SS	1880
Year-to-Date Commission	000000CCCC‸CC	1881
Current Sales	000000AAAA‸AA	1882

The salesman's basic commission is 5% of sales with an extra 2% for total sales in excess of $50,000. Update the year to date sales and commission, and at the point where year to date sales exceed $20,000 print

ΔQUOTAΔMET.Δ

7. Reading the data stores

DATA	FORM	CELL
Inventory Quantity	000000QQQQQQ‸	1880
Sales Quantity	000000SSSSSS‸	1881
Minimum Requirements	000000RRRRRR‸	1882

Update the inventory. If the inventory quantity falls below the minimum requirements, print the quantity needed to restore the inventory to its minimum level. This quantity is to be in form

000000PPPPPP‸

SUMMARY

A flow chart is a method for efficient logical analysis of computer applicat:ons and consists of

1. a set of symbols

and 2. rules for organizing the symbols.

The symbols are as follows.

A directed line segment

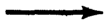


FIGURE 4-13

indicates the logical line of flow. The start symbol

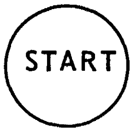


FIGURE 4-14

indicates the beginning of the logical line of flow; the stop symbol,

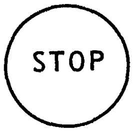


FIGURE 4-15

the end of the logical line of flow. Transfer and arithmetic operations are shown in rectangles;

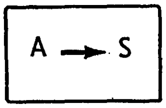


FIGURE 4-16

decisions, in flattened ovals.

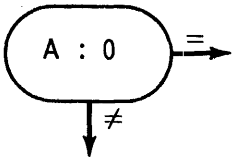


FIGURE 4-17

Connectors

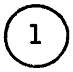


FIGURE 4-18

indicate the merging of divergent lines of flow.

chapter 5

Modification of Instructions

Both data and instructions are stored in the memory. The computer recognizes an instruction as such only when it is in SR. At no other time does the computer make distinction between data and instructions. Both are simply words stored in the memory. This arrangement enables a word which has been interpreted as instructions at one time in a program to be processed as data by other instructions in the same program, thus allowing the computer to modify its own instructions. The following is an example of the modification of instructions.

CODING

```
0000    500003
                  ⌣
0001    B00000
                  A00005
0002    C00000
                  U00000
------------------------
0003    ΔΔELEC              \
                  TRONIC     |
0004    ΔCOMPU               |  Constants
                  TER.ΔΔ     |
0005    000001               |
                  900000    /
```

The execution of this coding will print:

ELECTRONIC COMPUTER.

and stop the computer.

First three stage cycle

β Time - The contents of cell 0000

500003000000.

are transferred to CR.

γ Time The LHI

500003

is transferred to SR and executed, printing the contents of cell 0003:

ELECTRONIC

δ Time - The RHI

000000

is transferred to SR and executed, skipping to the next stage.

Second three stage cycle

β Time - The contents of cell 0001

B00000A00005

are transferred to CR.

γ Time - The LHI

B00000

is transferred to SR and executed, transferring the contents of the cell specified, cell 0000,

500003000000

to rA and rX. This word is treated as an instruction pair only when it is in CR; at all other times it is treated as data or a constant. This word, which was treated as an instruction during the first three stage cycle, is now treated as data being processed by an instruction in SR.

δ Time - The RHI

A00005

is transferred to SR and executed, transferring the contents of cell 0005 to rX, adding the contents of

```
         rA: 500003000000
and      rX: 000001900000
-------------------------
```

and transferring the sum

500004900000

to rA.

Third three stage cycle

β Time - The contents of cell 0002

C00000U00000

are transferred to CR.

γ Time - The LHI

C00000

is transferred to SR and executed, transferring the contents of rA:

5000004900000

to the cell specified, cell 0000.

δ Time - The RHI

U00000

is transferred to SR and executed, transferring control to cell 0000.

Fourth three stage cycle

β Time - The contents of cell 0000, which now contains the word

500004900000

are transferred to CR.

γ Time - The LHI

500004

is transferred to SR and executed, printing

COMPUTER

δ Time - The RHI

900000

is transferred to SR and executed, stopping the computer.

On δ Time of the second three stage cycle the computer added a positive 11 digit quantity

000001900000

to an 11 digit quantity with a five in the sign position

500003000000

to arrive at an eleven digit sum with a five in the sign position

500004900000

This sum resulted because of the following characteristics of the adder.

Of two words to be added at least one must have an actual sign, 0 or -, in the sign position. If neither has a sign, the computer stalls and lights a neon on the Supervisory Control Panel, thus indicating that an error, called an adder-alphabetic error, has occurred.

For purposes of the addition, any character in the sign position other than a minus sign is treated as a plus sign, when the other word to be added has a legitimate sign. For example, the character A would be treated as a plus sign. When the sum is transferred to rA, the sign position will contain, not the sign of the sum, but the character A. In any digit position other than the sign position, the addition of

1. two numbers produces an algebraic sum,
2. a number and an alphabetic produces the alphabetic,
3. two alphabetics produces an adder alphabetic error.

ITERATIVE CODING

Example

Reading the data stores a credit account number of form

AAAAAAAAAAAA

in cell 1820, and 60 delinquent account numbers of form

DDDDDDDDDDDD

in cells 1880 - 1939. If the credit account number is equal to one of the delinquent account numbers print

ΔNOΔCREDIT.Δ

if not,

CREDITΔGOOD.

FLOW CHART

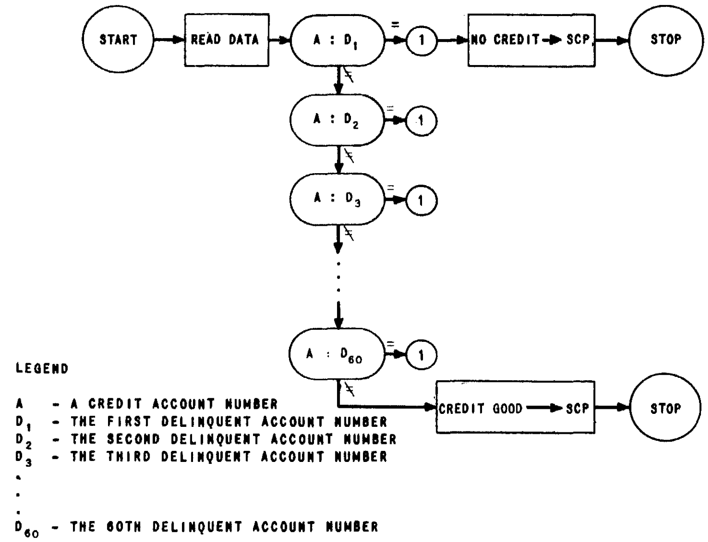

START
READ DATA
A : D_1
=
1
NO CREDIT → SCP
STOP
≠
A : D_2
=
1
≠
A : D_3
=
1
≠
A : D_{60}
=
1
≠
CREDIT GOOD → SCP
STOP
LEGEND
A - A CREDIT ACCOUNT NUMBER
D_1 - THE FIRST DELINQUENT ACCOUNT NUMBER
D_2 - THE SECOND DELINQUENT ACCOUNT NUMBER
D_3 - THE THIRD DELINQUENT ACCOUNT NUMBER
D_{60} - THE 60TH DELINQUENT ACCOUNT NUMBER

FIGURE 5-1

```
CODING 0000  READ                 }
                         DATA     }   read data
       0001  L01820               }
                         ↪        }   A : D1
       0002  B01880               }
                         Q00063   }
       ------------------------
       0003  B01881               }   A : D2
                         Q00063   }
       ------------------------
       0004  B01882               }   A : D3
                         Q00063   }
       ------------------------
         •
         •
         •
       0061  B01939               }   A : D60
                         Q00063   }
       ------------------------
       0062  500064                   CREDIT GOOD → SCP
                         900000       stop
       ------------------------
  (1)  0063  500065                   NO CREDIT → SCP
                         900000       stop
       ------------------------
       0064  CREDIT               }
                         ΔGOOD.   }
       0065  ΔNOΔCR               }   constants
                         EDIT.Δ   }
```

The coding shows that each delinquent account number is processed the same way. The coding to process one delinquent account number, after executing the L01820, takes the form

BOXXXXQ00063

where XXXX is the address of the delinquent account number being processed. Since there are 60 delinquent account numbers to be processed, and since each delinquent account number is in a different cell, the above instructions are repeated 60 times. However, the above instructions can be stored only once and can be used to process all 60 delinquent account numbers by modifying the address specified by the BOm instruction and transferring control to repeat the processing.

```
0000  READ                 }
                  DATA     }   read data
0001  B01880               }
                  L01820   }   Does the credit account number match
0002  ↪                    }   the current delinquent account number?
                  Q00008   }
------------------------
```

```
0003   B00001                        B01880L01820→rA
                     ↪
0004   ↪                        Take the next delinquent account number.
                     ↪
0005   A00010                        add    000001 000000
                     C00001                 B01881 L01820→0001
0006   ↪
                     U00001
-------------------------
0007   ↪
                     ↪
0008   500012                   NO CREDIT → SCP
                     900000     stop
-------------------------
0009   ↪
                     ↪
0010   000001
                     ↪
0011   ↪                        constants
                     ↪
0012   ΔNOΔCR
                     EDIT.Δ
```

This coding allows the credit account number to be compared to the delinquent account numbers in succession as long as there is inequality. If the credit account number is not one of the delinquent account numbers, cell 0001 will eventually contain the instruction pair

B01939L01760

After each iteration the contents of cell 0001 can be compared for identity with the above word. This comparison determinès the end of the processing, much as a student reading an assignment might check each page number to see if he has completed the assignment.

```
0000   READ                     read data
                     DATA
0001  [B01880
                     L01820]    Does the credit account number match
0002   ↪                        the current delinquent account number?
                     Q00008
-------------------------
```

```
0003  B00001
                 L00009      Is the current delinquent account number
0004  ↪                      the last delinquent account number?
                 Q00007
-------------------------
0005  A00010                 Take the next delinquent account number.
                 C00001
0006  ↪
                 U00001
-------------------------
0007  500011                 CREDIT GOOD → SCP
                 900000      stop
-------------------------
0008  500012                 NO CREDIT → SCP
                 900000      stop
-------------------------
0009  B01939
                 L01820
0010  000001     ↪
                             constants
0011  CREDIT
                 ΔGOOD.
0012  ΔNOΔCR
                 EDIT.Δ
```

By custom, lines of coding that are subject to alteration are enclosed in brackets to distinguish them from lines which do not vary. This custom is of help in checking coding for correctness, both before and after it is run on the computer.

The principle shown in this example is called iterative coding.

Care must be taken in stopping the iteration at the right time. In the coding on page 91 the constant used to determine if all delinquent account numbers have been processed is

B01939L01820

In the following coding the constant is

B01940L01820

```
0000  READ
                 DATA        read data
0001 [B01880       ]
     [           L01820]     Does the credit account number match
0002  ↪                      the current delinquent account number?
                 Q00007
-------------------------
```

```
0003  B00001
                  L00008     Is the current delinquent account number
0004  A00009                 the last delinquent account number?
                  Q00006
0005  C00001                 Take the next delinquent account number.
                  U00001
0006  500010                 CREDIT GOOD → SCP
                  900000     stop
0007  500011                 NO CREDIT  → SCP
                  900000     stop
0008  B01940
                  L01820
0009  000001
                             constants
0010  CREDIT
                  ΔGOOD.
0011  ΔNOΔCR
                  EDIT.Δ
```

The reason for the difference in constants is that,in the coding on page 91, the execution of the QOm instruction, which determines if all delinquent account numbers have been processed, precedes the execution of AOm instruction, which alters the address to process the next delinquent account number; while in the coding on page 92, the execution of the AOm instruction precedes the execution of the QOm instruction.

Item Just Processed	(rA) During Execution of QOm Instruction	
	In Coding on Page 91	In Coding on Page 92
1st	B01880L01820	B01881L01820
2nd	B01881L01820	B01882L01820
3rd	B01882L01820	B01883L01820
.	.	.
.	.	.
.	.	.
58th	B01937L01820	B01938L01820
59th	B01938L01820	B01939L01820
60th	B01939L01820	B01940L01820

Iterative coding conserves memory space in that fewer instructions need be stored in the memory to do the processing.

The memories of computers are limited in capacity because of the high costs for memory per digit stored. Consequently, the more processing that can be done per instruction stored, the greater is the area of the memory freed for the storage of data and other instructions. Iterative coding is a powerful technique in the efficient programming of computers.

ITERATIVE FLOW CHART SYMBOLS

In a word flow chart, the solution might appear as:

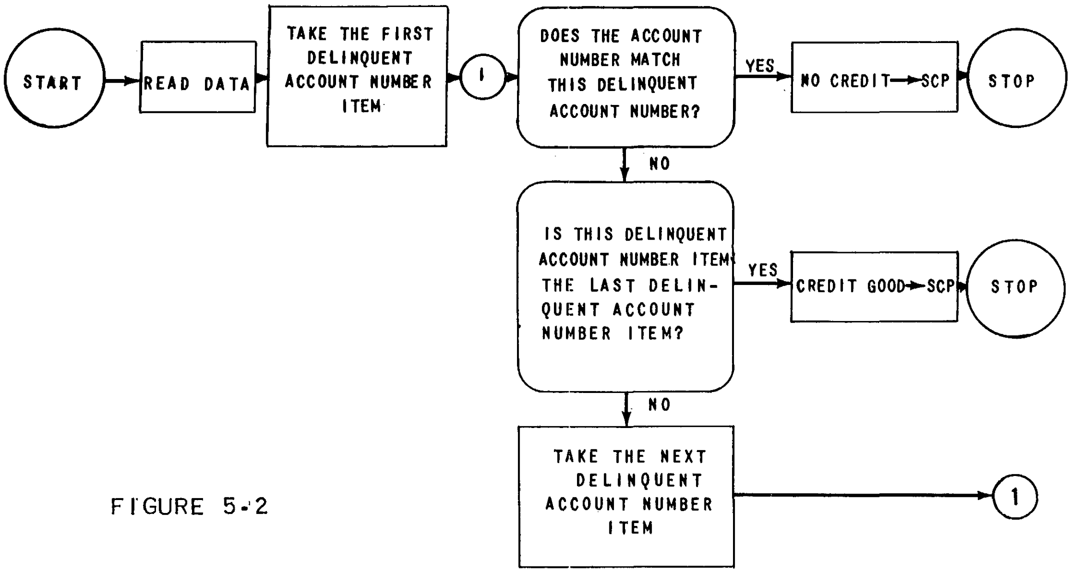

START
READ DATA
TAKE THE FIRST DELINQUENT ACCOUNT NUMBER ITEM
1
DOES THE ACCOUNT NUMBER MATCH THIS DELINQUENT ACCOUNT NUMBER?
YES
NO CREDIT → SCP
STOP
NO
IS THIS DELINQUENT ACCOUNT NUMBER ITEM THE LAST DELINQUENT ACCOUNT NUMBER ITEM?
YES
CREDIT GOOD → SCP
STOP
NO
TAKE THE NEXT DELINQUENT ACCOUNT NUMBER ITEM
1

FIGURE 5-2

A set of data is represented by a capital letter. The set of delinquent account numbers might be represented as D.

To distinguish between units in a set, numeric subscripts are used. In the set D

D_1 represents the first delinquent account number.

D_2 represents the second delinquent account number.

D_3 represents the third delinquent account number.

.

.

.

D_{60} represents the 60th delinquent account number.

Only one unit in a set is processed at a time and may be identified by an alphabetic subscript. For example, D_i might represent the delinquent account number currently being processed from the set D. The alphabetic subscript is used because, although only one unit is processed at a time, it cannot be stated specifically which unit is being processed at a given time.

Units are processed sequentially. Unit D_1 is processed first, D_2, second; D_3, third; etc. In general, after unit D_i has been processed, unit D_{i+1} is to be processed. The operation

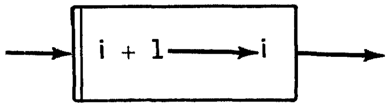


FIGURE 5-3

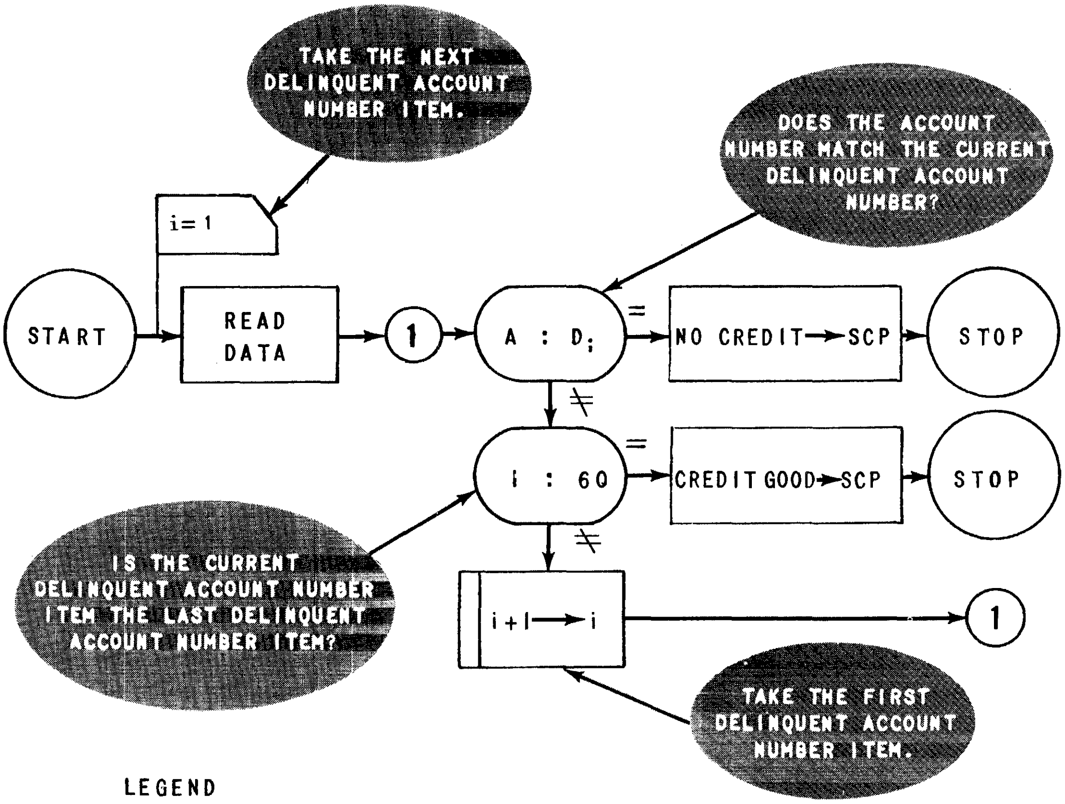
provides this sequence. The operation box has a double line on the left to distinguish it from an operation which processes data. The initial condition for the sequence is that i be equal to one so that $D_i = D_1$. Initial conditions of the processing are shown in an assertion flag placed immediately after the start symbol.

ILLUSTRATIVE EXAMPLE

Reading the data stores 60 receipt amounts of form

000000RRRR‸RR

in cells 1880-1939. Print the sum of the amounts.


TAKE THE NEXT DELINQUENT ACCOUNT NUMBER ITEM.
i= 1
DOES THE ACCOUNT NUMBER MATCH THE CURRENT DELINQUENT ACCOUNT NUMBER?
START
READ DATA
1
A : Dᵢ
=
NO CREDIT → SCP
STOP
≠
i : 60
=
CREDIT GOOD → SCP
STOP
≠
IS THE CURRENT DELINQUENT ACCOUNT NUMBER ITEM THE LAST DELINQUENT ACCOUNT NUMBER ITEM?
i+1 → i
1
TAKE THE FIRST DELINQUENT ACCOUNT NUMBER ITEM.

LEGEND

A AN ACCOUNT NUMBER

D A SET OF DELINQUENT ACCOUNT NUMBER ITEMS

D_i THE iTH ITEM IN D, i = 1, . . . , 60

FIGURE 5-4

FLOW CHART

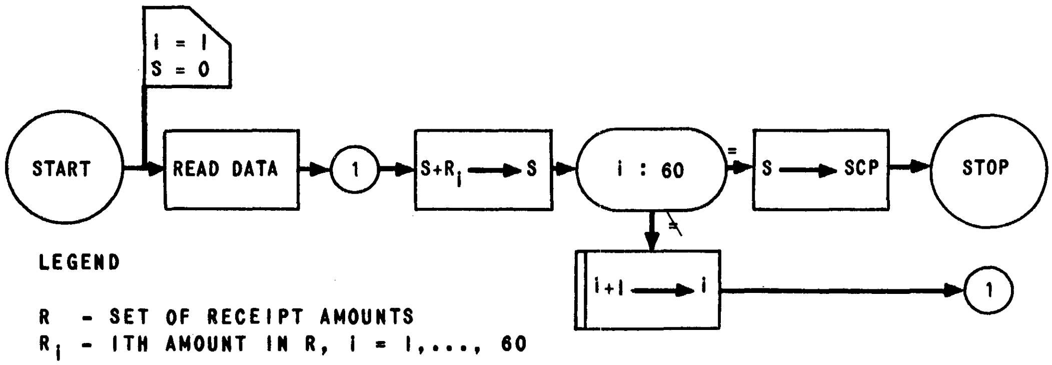


FIGURE 5-5

The following is a description of the thinking that might have accompanied the flow chart. The coding is on page 22.

Flow chart the general processing

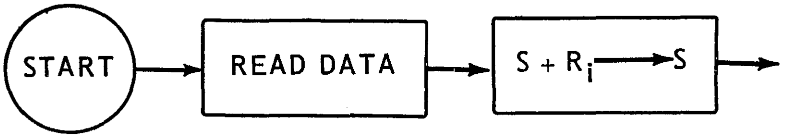


FIGURE 5-6

Specify the general. Initially, i is equal to one; and the sum, equal to zero.

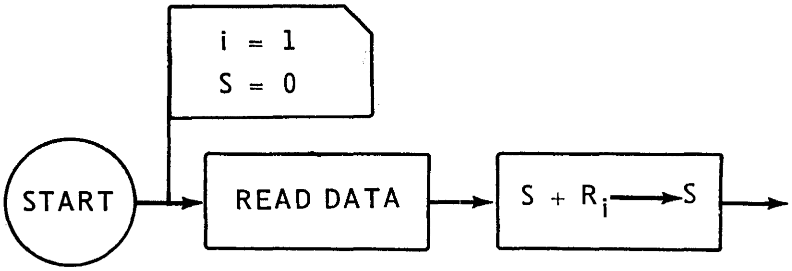


FIGURE 5-7

After the first amount is processed, the computer should advance to the second amount.

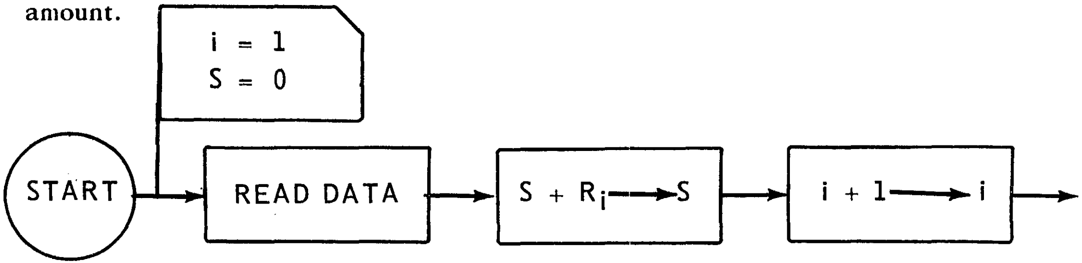


FIGURE 5-8

The second amount should be processed in the same way as the first.

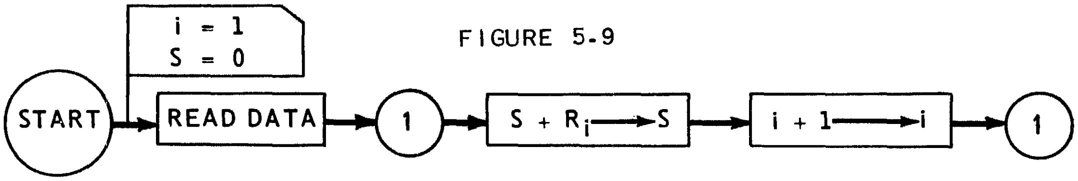


FIGURE 5-9

Thus, specifying the general sets up the iterative loop.

Finally, providing the exit from the iterative loop and flow charting the ending routine completes the flow chart, which is shown in figure 5-5.

```
   0000   READ             }
                  DATA     }   read data
①  0001  [B01880          ]}
                  A00007  ]}   S + Ri → S
   0002   C00007           }
                  B00001
   0003   L00008           }
                  Q00006   }   i : 60
   ---------------------
   0004   A00009           }
                  C00001   }   i + 1 → i
   0005   ⊂⊃
                  U00001
   ---------------------
   0006   500007               S → SCP
                  900000       stop
   ---------------------
   0007  [⊂⊃      ⊂⊃      ]}   S
   0008   B01939           }
                  A00007   }   constants
   0009   000001           }
                  ⊂⊃
```

ARITHMETIC INSTRUCTIONS - LIST C

The function of certain pairs of instructions have been combined into single instructions, resulting in a saving of computer time and memory space. One such instruction, the AHm instruction, has a powerful application in iteration and summation.

INSTRUCTION	OPERATION	MNEMONIC
AHm	(m) → rX; (rA) +(rX) → rA; (rA) → m	Add and Hold

Add (m) to (rA). Transfer the sum to m, and hold the result in rA.

The mnemonic is to add (m) to (rA) and hold the sum in rA, after transferring it to m. The computer executes the AHm instruction as follows. (m) are transferred to rX. (rA) and (rX) are added. The sum is transferred to rA. (rA) are transferred to m.

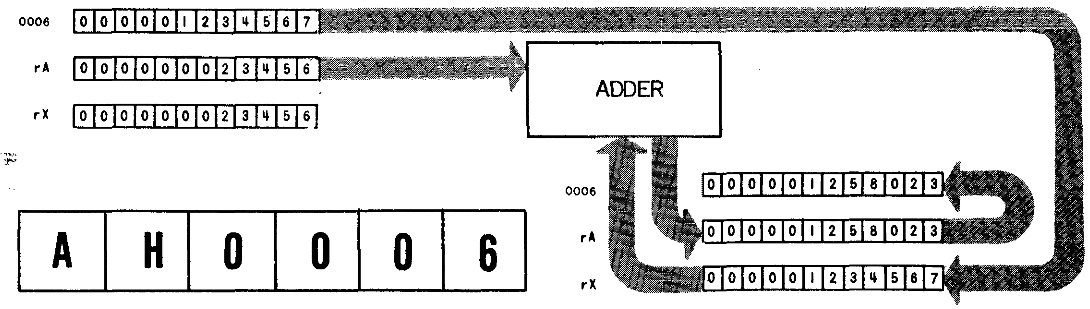


FIGURE 5-10

One of the possible uses of the AHm instruction is to add a quantity to the contents of a cell. Suppose that the contents of cell 0007 are to be added to the contents of cell 0003. Without the use of the AHm instruction the coding might be

```
0001     B00003
                      A00007
0002     C00003
```

Using the AHm instruction

```
0001     B00007
                      AH0003
```

The AHm instruction is used to reduce the number of instructions needed to do a processing operation. This reduction is achieved because the AHm instruction combines in one instruction the function of the AOm and HOm instructions. Another saving is in the reduction of the computer time required to perform the operations.

INSTRUCTION	OPERATION	MNEMONIC
SHm	$-(m) \longrightarrow rX;\ (rA)+(rX) \longrightarrow rA;\ (rA) \longrightarrow m$	Subtract & Hold

Subtract (m) from (rA). Transfer the difference to m.

The mnemonic is to subtract (m) from (rA) and hold the difference in rA. The computer executes the SHm instruction the same way as it executes the AHm instruction except that minus (m) are transferred to rX.

Using the AHm instruction, the coding for the flow chart in figure 5-5 might be as follows.

```
  0000   READ                 read data
                DATA
① 0001   B00007
                AH0003        i : 60        m→rX; (rA) + (rX)→rA; (rA)→m
  0002   L00008
                Q00005
  0003 [ B01878
                A01879 ]      S + R_i→S
  0004   AH0006                             m→rX; (rA) + (rX)→rA; (rA)→m
                U00001
  0005   500006               S→SCP
                900000        stop
  0006 [                ]     S
  0007   000002
                000002        constants
  0008   B01940
                A01941
```

The steps in this coding and the steps in the flow chart are not in the same sequence, and the summation step is not even the same. However, the two approaches are logically equivalent and lead to the same result. It is not necessary that the coding steps follow the flow chart slavishly; it is only required that the coding accomplish the same purpose as the flow chart. Neither is it necessary that the flow chart reflect in complete detail the steps in the coding. Actually, such a situation is generally impossible since at the flow charting stage, it is not likely that the exact steps in the coding can be predicted. The only requirement is that both the flow chart and the coding solve the problem - the flow chart, logically; the coding, on the computer. If a flow chart were to reflect the above coding, it would be as follows.

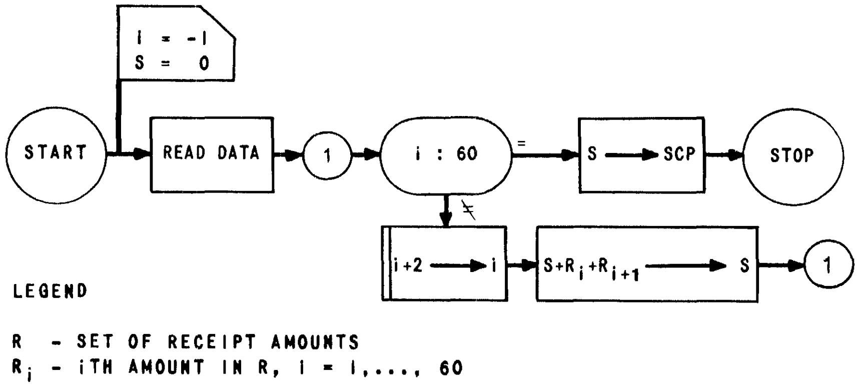


FIGURE 5-11

STUDENT EXERCISES

1. Reading the data stores 60 quantities of form

$$\pm \mathrm{QQQQQQQQQQ}_{\wedge}$$

in cells 1880 - 1939. Print the number of negative quantities.

2. Reading the data stores

1. a pay of form

$$\mathrm{0000_{\wedge}00PPPPPP}$$

in cell 1880

and 2. ten deductions of form

$$\mathrm{0000_{\wedge}0000DDDD}$$

in cells 1881 - 1890.

Each deduction is processed as follows. If the deduction will not reduce the pay below $15, it is applied. If the deduction will reduce the pay below $15, it is not applied but is printed instead. When all deductions have been processed, print the pay.

3. Reading the data stored 60 quantities of form.

$$\mathrm{000000QQQQQQ}_{\wedge}$$

in cells 1880 - 1939. Print the subtotal of each group of ten quantities and the total of the quantities.

FUNCTION TABLE LOOK-UP

ILLUSTRATIVE EXAMPLE

Reading the data stores

1. an employee's base pay of form

$$\mathrm{00000BBBB_{\wedge}BB0}$$

in cell 1880,

2. the employee's shift of form

   00000000000S

   in cell 1881, where S is a key and can take values 1-6,

3. six percentages of form

   0‸PPP00000000

   in cells 1821 - 1826.

The employee is paid a shift differential, each shift drawing a different percentage of base pay. The shifts and the cells in which the applicable percentages are stored are in the following relationship.

SHIFT	CELL
1	1821
2	1822
3	1823
4	1824
5	1825
6	1826

Print the pay in form

000000AAAA‸AA

The problem could be solved by testing the shift key against each possible value, and on the basis of the tests, choosing the appropriate percentage. However, if this approach were used, the majority of the coding would be concerned, not with the problem of computing the pay, but with choosing the appropriate percentage, which is merely preparatory to the problem solution. The following approach eliminates this disadvantage. The table in the example shows that the shift key is in a one to one relationship with the units digit of the address of the cell in which the appropriate percentage is stored. If S represents the shift key; and m, the address of the appropriate cell; the following holds.

$$m = 1820 + S$$

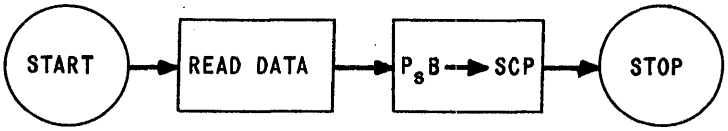
This relationship, or function, can be used to derive the appropriate cell directly from the shift key.

```
0000   READ                      read data
                 DATA
0001   B01881
                 A00006          Derive cell from shift.
0002   C00003
0003   L01880
                 M0182S          Print pay.
0004   C01882
                 501882
0005   900000                    stop
0006   L01880                    constant
                 M01820.
```

Since this coding uses a function to look up the appropriate percentage from a table, it is an example of the technique called "function table look-up". Function table look-up is a programming principle that makes use of a relationship between the data and the addresses of the cells in which the data is stored to increase computer efficiency with respect to the conservation of both memory space and computer time.

FUNCTION TABLE LOOK-UP IN FLOW CHARTS

If a capital letter is used to represent the table, the table entries can be represented by subscripts. The entry desired depends on the argument with which the table is entered. If P represents the percentage table in the above example; and S, the shift key; the entry desired can be represented as P_S.



LEGEND

S - SHIFT
P - A SET OF PERCENTAGES
P_i - THE ITH PERCENTAGE IN P, i = 1,...., 6
B - BASE PAY

FIGURE 5-12

SHIFT INSTRUCTIONS

n is used to represent a variable second instruction digit.

INSTRUCTION	OPERATION
Onm	Shift (rA), excluding sign, n position left.

With the exception of the sign digit, shift (rA) left n digit positions. Transfer zeros to the vacated positions.

When executing the Onm instruction, the computer ignores m. Characters shifted beyond the capacity of rA are lost.

INSTRUCTION	OPERATION
– nm	Shift (rA), excluding sign, n positions right.

With the exception of the sign digit, shift (rA) right n digit positions. Transfer zeros to the vacated positions.

When executing the -nm instruction, the computer ignores m. Characters shifted beyond the capacity of rA are lost.

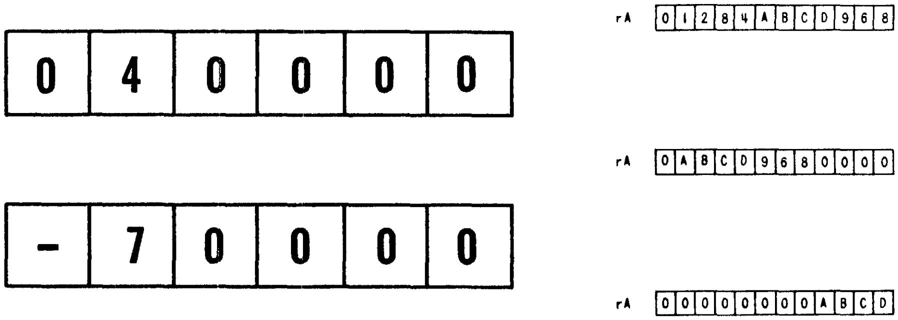

rA 0 1 2 8 4 A B C D 9 6 8
0 4 0 0 0 0
rA 0 A B C D 9 6 8 0 0 0 0
- 7 0 0 0 0
rA 0 0 0 0 0 0 0 0 A B C D

FIGURE 5-13

INSTRUCTION	OPERATION
;nm	Shift (rA) left n positions

Shift (rA) left n digit positions. Transfer zeros to the vacated positions.

When executing the ;nm instruction, the computer ignores m. Charcters shifted beyond the capacity of rA are lost.

INSTRUCTION	OPERATION
.nm	Shift (rA) n positions right.

Shift (rA) right n digit positions. Transfer zeros to the vacated positions.

When executing the .nm instruction, the computer ignores m. Characters shifted beyond the capacity of rA are lost.

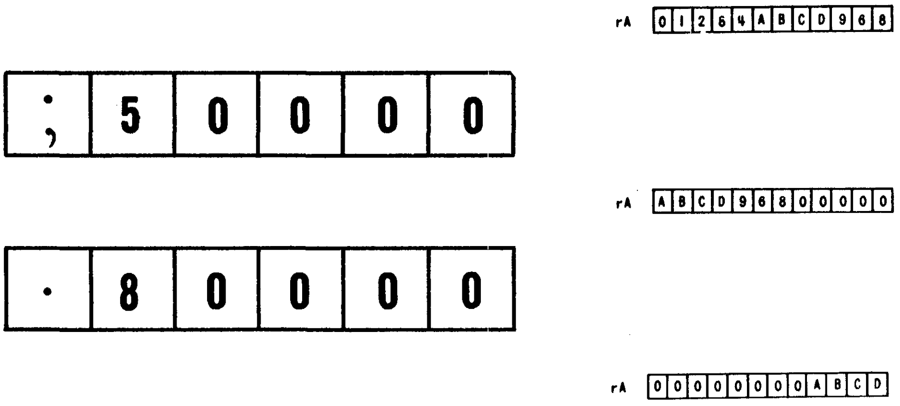

rA 0 1 2 6 4 A B C D 9 6 8
; 5 0 0 0 0
rA A B C D 9 6 8 0 0 0 0 0
· 8 0 0 0 0
rA 0 0 0 0 0 0 0 0 A B C D

FIGURE 5-14

EXAMPLE

Reading the data stores

1. the weight, in pounds, of a package of form

   00000WWW$\underset{\wedge}{\text{W}}$000

   in cell 1820

and 2. 60 shipping rates in dollars and cents per pound, of form

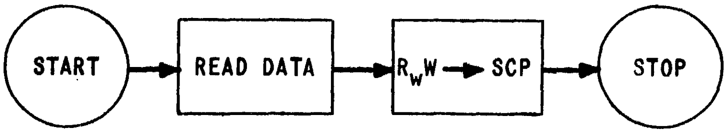
   0$\underset{\wedge}{\text{R}}$RR00000000

   in cells 1900 - 1959.

For WEIGHT	apply rate stored in CELL
0000 - 0099	1990
0100 - 0199	1901
0200 - 0299	1902
.	.
.	.
.	.
5900 - 5999	1959

Print the cost to ship the package.

FLOW CHART



LEGEND

W - WEIGHT
R - A SET OF RATE ITEMS
R_i - THE iTH ITEM IN R, i = 1,..., 60

FIGURE 5-15

CODING

The table in the example shows that the thousands and hundreds digits of the weight are in a one to one relationship with the tens and units digits of theaddress of the cell in which the appropriate rate is stored. Thus, the address of the appropriate cell is 1900 added to the two most significant digits of the weight. Dividing the weight by 100 will give the quantity to be added to 1900, since the table intervals are 100 pounds. However, since the weight may not be multiple of 100, the quotient may also contain a fractional part. If the weight were 4627,

$$\frac{4627}{100} = \underbrace{46}_{\text{integral part}}.\underbrace{27}_{\text{fractional part}}$$

Thus if W is the weight, the quantity to be added to 1900 is the integral part of

$$\frac{W}{100}$$

represented by

$$\left(\frac{W}{100}\right)_{IP}$$

If m represents the appropriate cell, the function is

$$m = 1900 + \left(\frac{W}{100}\right)_{IP}$$

If the computer divides the weight by 100, both the integral and fractional parts of the quotient will be transferred to rA. The parts might be separated by use of a shift instruction.

In the following coding no divide instruction is actually used, since division by 100 can be performed by moving the assumed decimal point two positions to the left.

```
0000   READ              }   read data
               DATA      }
0001   B01820            )
               50000     |
0002   A00006            |
               C00003    |   Rw W → SCP
0003  [L01820           ]|
               M019WW   ]|
0004   C01821            |
               501821    )
0005   900000                stop
_____________________
               000000
0006   L01820            }   constant
               M01900    }
```

EXAMPLE

Reading the data stores

1. the weight in pounds of a package of form

$$0000WWWWWW_{\wedge}000$$

in cell 1820

and 2. 60 shipping rates per pound of form

$$0R_{\wedge}RR000000000$$

in cells 1900 - 1959

For	WEIGHT	apply rate stored in	CELL
	00000 - 00249		1900
	00250 - 00499		1901
	00500 - 00749		1902
	.		.
	.		.
	.		.
	14750 - 14999		1959

Print the cost to ship the package.

FLOW CHART   See Figure 5-15

CODING

If W represents the weight;and m, the appropriate cell; the function is

$$m = 1900 + \left(\frac{W}{250}\right)_{IP}$$

In the following coding a multiply rather than a divide instruction is used, because for a known number, it is always faster for the computer to multiply by the reciprocal of the number than to divide by the number itself.

```
0000   READ            }     read data
              DATA     }
0001   L01820          }
              P00006   }
0002   A00007          }
              C00003   }     R_W W --> SCP
0003   L01820          }
              M019WW   }
0004   C01821          }
              501821   }
0005   900000                stop
-------------------------
0006   000000          }
              400000   }     constants
0007   L01820          }
              M01900   }
```

STUDENT EXERCISES

1. Reading the data stores

DATA	FORM	CELC
Quantity A	±AAAAA‸000000	1880
Quantity B	±00000BBBBBB‸	1881

Print the sum of the quantities in form

+0000SSSSSSS‸

2. Reading the data stores three quantities, A, B and C, of form

AAAA‸BBBB‸CCCC‸

in cell 1880. Print the quantities, each in form

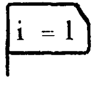
OOOOOOOOQQQQ‸

SUMMARY

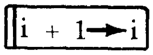
A word is only treated as an instruction pair if and when it is transferred to the Control Unit. At other times it has the same properties as all Univac words.

ITERATIVE CODING: the method of processing a set of units by modifying the instructions which refer to the first unit and repeating the instructions.

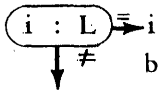
If Di represents the ith unit of set D:



the assertion flag indicates that the initial value of i is 1.



Select the next unit in the set



if i = 1, 2, 3, ... ,L; i = L indicates that the last unit has been processed.

FUNCTION TABLE LOOK-UP: the technique of locating an entry in a stored table by means of some function between the address of the entry and the entry. In a flow chart P_S might indicate the desired shift differential, S, in a table P.

Instructions

AHm: (m) → rX; (rA) + (rX) → rA, m
SHm: -(m) → rX; (rA) + (rX) → rA, m
.nm : shift (rA) right n places, including sign
-nm : shift (rA) right n places, excluding sign
;nm : shift (rA) left n places, including sign
0nm : shift (rA) left n places, excluding sign

chapter 6

Item Processing

THE ITEM

A unit of data is called an item. For example, each delinquent account number in the set of delinquent account numbers in the example on page 88 is a unit of data, or an item.

THE FIELD

Up to this point an item has been a single piece of information. In general, an item consists of more than one piece of information, called fields, and is generally composed of more than one word. An inventory item may contain at least the following fields.

1. Stock number.
2. Description
3. On hand quantity
4. On order quantity
5. Minimum requirements
6. Unit price

An inventory item might have the form

```
word 0:  NNNNNNNNNNNN
     1:  DDDDDDDDDDDD
     2:  0HHHHHHH0000
                 ^
     3:  000000000000
                 ^
     4:  0RRRRRRR0000
                 ^
     5:  0PPPPP000000
            ^
```

where N - Stock number
D - description
H - on hand quantity
O - on order quantity
R - minimum requirements
P - unit price

REPRESENTING FIELDS ON FLOW CHARTS

Fields are represented by superscripts to the item symbol. If I is the set of inventory items; and I_i, the ith item in I,

I_i^N is the stock number of I_i

I_i^D - the description of I_i

I_i^H - on hand quantity of I_i

I_i^O - on order quantity of I_i

I_i^R - minimum requirements of I_i

I_i^P - unit price of I_i

WRITING DATA

Up to this point problems have been such that the results of processing, or output data, have been small in quantity, and the SCP has been used to print the output data. Generally, output is large, and printing it directly from the computer would be inefficient, since a printer operates much more slowly than a computer.

Computer output is generally recorded on tape. There are instructions, called write instructions, which when executed, perform the writing. Write instructions will not be described here. Instead, writing data will be indicated by the words, "Write Data".

Just as it is generally inefficient to read input data an item at a time, it is generally inefficient for a computer to write output an item at a time. Instead, output items are grouped in the memory and are written on tape as a group.

ILLUSTRATIVE EXAMPLE

Reading the data stores, in cells 1880-1939, 12 five word job items of the form:

```
NNNNNNN00000
0000000CCCCC
          ^
0000000LLLLL
          ^
0000000MMMMM
          ^
0000000OOOOO
          ^
```

where N - job number
C - contract price
L - labor cost
M - material cost
O - overhead cost

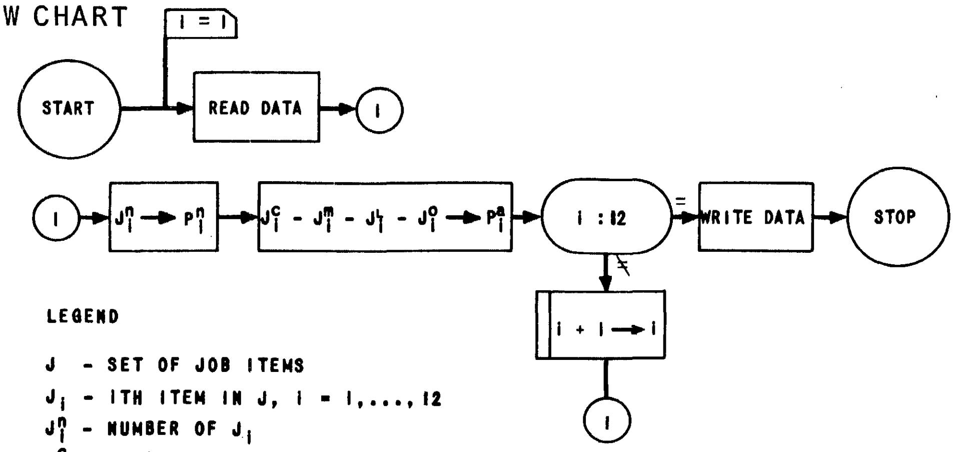
For each job item produce a two word profit item of form:

```
NNNNNNN00000
0000000AAAAA
          ^
```

where N - job number
A - profit

Write the profit items.

FLOW CHART


i = 1
START
READ DATA
1
1
$J_i^n \rightarrow P_i^n$
$J_i^c - J_i^m - J_i^L - J_i^o \rightarrow P_i^a$
i : 12
=
WRITE DATA
STOP
≠
i + 1 → i
1

LEGEND

J - SET OF JOB ITEMS

J_i - ITH ITEM IN J, i = 1,...,12

J_i^n - NUMBER OF J_i

J_i^c - PRICE OF J_i

J_i^m - MATERIAL COST OF J_i

J_i^L - LABOR COST OF J_i

J_i^o - OVERHEAD COST OF J_i

P - SET OF PROFIT ITEMS

P_i - ITH ITEM P, I = 1,...,12

P_i^n - NUMBER OF P_i

P_i^a - PROFIT OF P_i

FIGURE 6-1

CODING

	0000	READ	DATA	read data
(1)	0001	B01880	C01940	$J_i^N \rightarrow P_i^N$
	0002	B01881	S01882	$J_i^C - J_i^L - J_i^M - J_i^O \rightarrow P_i^A$
	0003	S01883	S01884	
	0004	C01941	B00001	
	0005	L00013	Q00011	i : 12
	0006	A00014	C00001	i + 1 → i
	0007	B00015	AH0002	
	0008	B00015	AH0003	
	0009	B00004	A00016	
	0010	C00004	U00001	

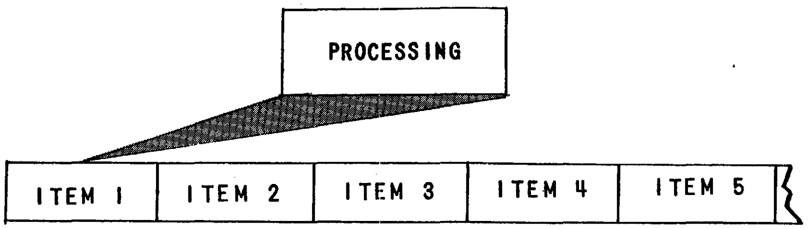
```
0011   WRITE                  write data
                DATA
0012   900000                 stop

0013   B01935
                C01962
0014   000005
                000002
0015   000005                 constants
                000005
0016   000002
```

WORKING STORAGE

A considerable portion of the above coding is composed of the instructions in cells 0004-0010, the instructions that alter the addresses of the processing instructions. This alteration is necessary so that after processing one item, the next will be processed. This set of instructions is called the item advance coding. The reason for the many instructions in the item advance coding is that each time an item is addressed by a processing instruction, that address must be modified to refer to the next item. The more an item is addressed in the processing, the longer the item advance coding will become. This disadvantage is removed by using working storage.

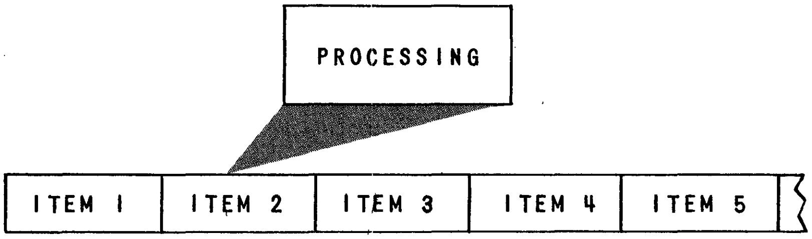
Using the previous method of item advance, the processing coding is initially directed toward the first item in the set.

FIGURE 6-2



When the first item has been processed, the direction of the processing is changed from the first to the second item.

FIGURE 6-3



When the second item has been processed the direction of the processing is changed to the third item, then the fourth item, etc.

A different approach to this problem is as follows. Initially the processing is directed toward the first item as shown in Figure 6-2. When the first item has been processed, instead of changing the direction of the processing to the second item, the second item is transferred to the location of the first.

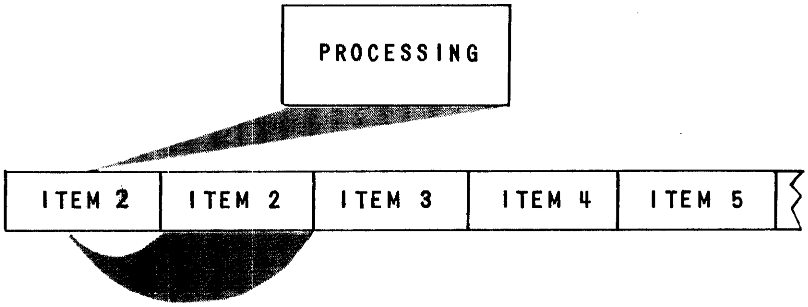


FIGURE 6-4

Thus, the second item can be processed with the same set of instructions. When the second item has been processed the third item is transferred to the first item location, etc.

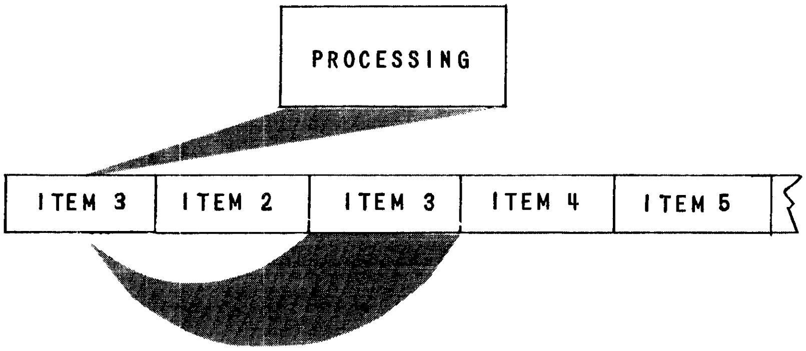


FIGURE 6-5

The area of the memory toward which the processing is directed is called working storage, since it is the area in which the item being processed.

The area in which items to be processed are stored is called the input area; the area in which the items resulting from processing are stored, the output area. Although working storage areas can be independent of the input and output areas, this situation is not necessarily the case. The first item location in the input area and the last item location in the output area are generally available for use as working storage areas, and for conservation of memory space, these locations are generally used.

By using working storage, the number of times an item is addressed in the processing has no affect on the amount of item advance coding. This amount is small, since it takes few instructions to move an item to working storage.

ITEM REGISTERS

To facilitate the movement of items, the computer has two item registers: register W [rW], which has a nine word capacity, and register Z [rZ], which has a 60 word capacity. Register W is made up of nine cells, each of which has a one word capacity. Register Z is made up of 60 such cells.

INSTRUCTION	OPERATION
Vnm	(m, ... , m + n — 1) ⟶ rW

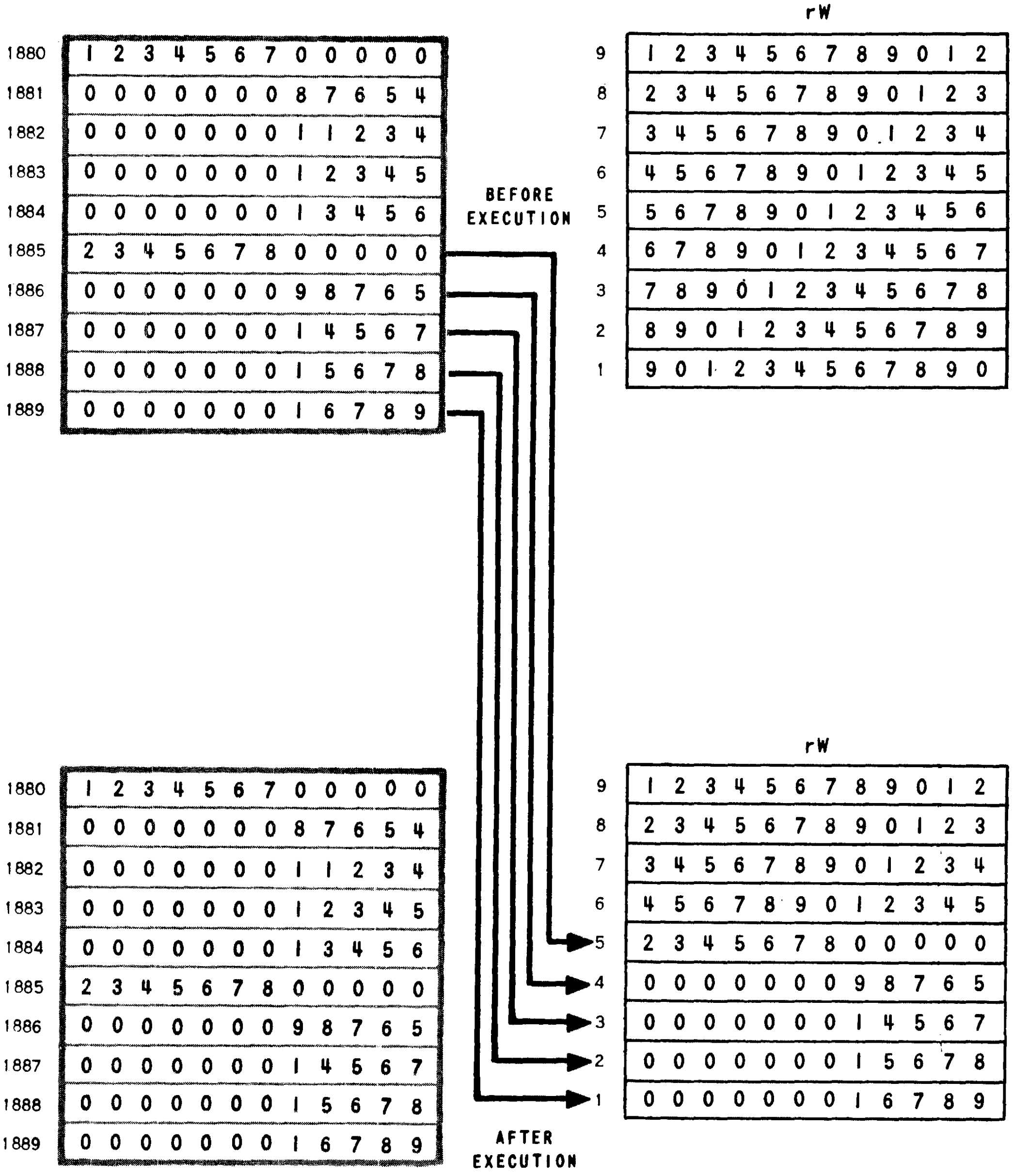
Transfer the contents of n consecutive memory cells, starting with m, to the first n cells of rW.

INSTRUCTION	OPERATION
Wnm	(rW) ⟶ m, ..., m + n — 1

Transfer the contents of the first n cells of rW to consecutive memory cells starting with m.

If the execution of a Vnm instruction is followed by the execution of a Wnm instruction, and if the second instruction digit is the same in both instructions, the execution of the Wnm instruction will store the words in the same sequence that the execution of the Vnm instruction selected them. The cases where the second instruction digit is not the same in each instruction is explained in a later chapter.

V	5	1	8	8	5



BEFORE EXECUTION

1880	1 2 3 4 5 6 7 0 0 0 0 0
1881	0 0 0 0 0 0 0 8 7 6 5 4
1882	0 0 0 0 0 0 0 1 1 2 3 4
1883	0 0 0 0 0 0 0 1 2 3 4 5
1884	0 0 0 0 0 0 0 1 3 4 5 6
1885	2 3 4 5 6 7 8 0 0 0 0 0
1886	0 0 0 0 0 0 0 9 8 7 6 5
1887	0 0 0 0 0 0 0 1 4 5 6 7
1888	0 0 0 0 0 0 0 1 5 6 7 8
1889	0 0 0 0 0 0 0 1 6 7 8 9

	rW
9	1 2 3 4 5 6 7 8 9 0 1 2
8	2 3 4 5 6 7 8 9 0 1 2 3
7	3 4 5 6 7 8 9 0 1 2 3 4
6	4 5 6 7 8 9 0 1 2 3 4 5
5	5 6 7 8 9 0 1 2 3 4 5 6
4	6 7 8 9 0 1 2 3 4 5 6 7
3	7 8 9 0 1 2 3 4 5 6 7 8
2	8 9 0 1 2 3 4 5 6 7 8 9
1	9 0 1 2 3 4 5 6 7 8 9 0

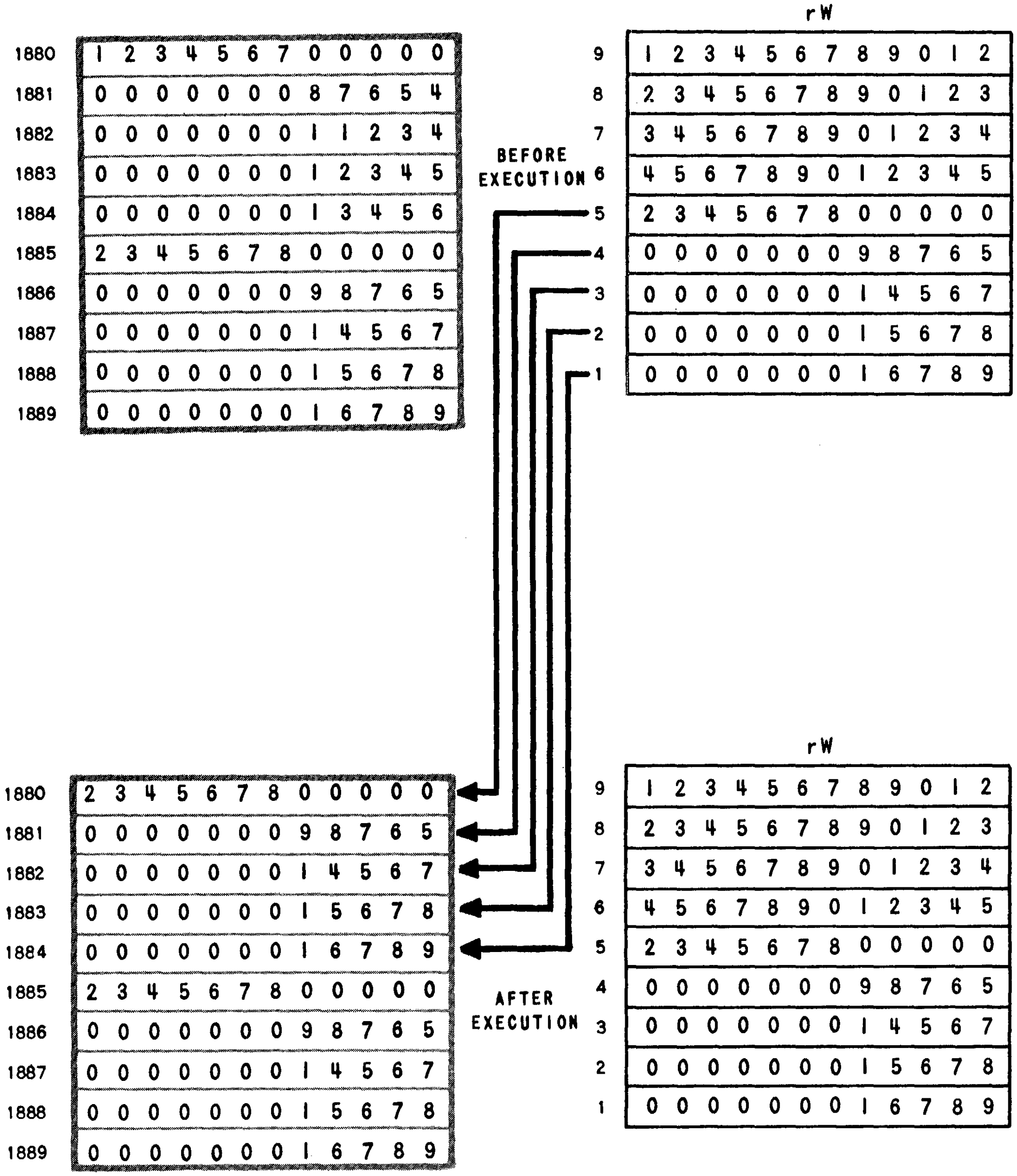
AFTER EXECUTION

1880	1 2 3 4 5 6 7 0 0 0 0 0
1881	0 0 0 0 0 0 0 8 7 6 5 4
1882	0 0 0 0 0 0 0 1 1 2 3 4
1883	0 0 0 0 0 0 0 1 2 3 4 5
1884	0 0 0 0 0 0 0 1 3 4 5 6
1885	2 3 4 5 6 7 8 0 0 0 0 0
1886	0 0 0 0 0 0 0 9 8 7 6 5
1887	0 0 0 0 0 0 0 1 4 5 6 7
1888	0 0 0 0 0 0 0 1 5 6 7 8
1889	0 0 0 0 0 0 0 1 6 7 8 9

	rW
9	1 2 3 4 5 6 7 8 9 0 1 2
8	2 3 4 5 6 7 8 9 0 1 2 3
7	3 4 5 6 7 8 9 0 1 2 3 4
6	4 5 6 7 8 9 0 1 2 3 4 5
5	2 3 4 5 6 7 8 0 0 0 0 0
4	0 0 0 0 0 0 0 9 8 7 6 5
3	0 0 0 0 0 0 0 1 4 5 6 7
2	0 0 0 0 0 0 0 1 5 6 7 8
1	0 0 0 0 0 0 0 1 6 7 8 9

FIGURE 6-6

W	5	1	8	8	0



BEFORE EXECUTION

1880	1 2 3 4 5 6 7 0 0 0 0 0
1881	0 0 0 0 0 0 0 8 7 6 5 4
1882	0 0 0 0 0 0 0 1 1 2 3 4
1883	0 0 0 0 0 0 0 1 2 3 4 5
1884	0 0 0 0 0 0 0 1 3 4 5 6
1885	2 3 4 5 6 7 8 0 0 0 0 0
1886	0 0 0 0 0 0 0 9 8 7 6 5
1887	0 0 0 0 0 0 0 1 4 5 6 7
1888	0 0 0 0 0 0 0 1 5 6 7 8
1889	0 0 0 0 0 0 0 1 6 7 8 9

	rW
9	1 2 3 4 5 6 7 8 9 0 1 2
8	2 3 4 5 6 7 8 9 0 1 2 3
7	3 4 5 6 7 8 9 0 1 2 3 4
6	4 5 6 7 8 9 0 1 2 3 4 5
5	2 3 4 5 6 7 8 0 0 0 0 0
4	0 0 0 0 0 0 0 9 8 7 6 5
3	0 0 0 0 0 0 0 1 4 5 6 7
2	0 0 0 0 0 0 0 1 5 6 7 8
1	0 0 0 0 0 0 0 1 6 7 8 9

AFTER EXECUTION

1880	2 3 4 5 6 7 8 0 0 0 0 0
1881	0 0 0 0 0 0 0 9 8 7 6 5
1882	0 0 0 0 0 0 0 1 4 5 6 7
1883	0 0 0 0 0 0 0 1 5 6 7 8
1884	0 0 0 0 0 0 0 1 6 7 8 9
1885	2 3 4 5 6 7 8 0 0 0 0 0
1886	0 0 0 0 0 0 0 9 8 7 6 5
1887	0 0 0 0 0 0 0 1 4 5 6 7
1888	0 0 0 0 0 0 0 1 5 6 7 8
1889	0 0 0 0 0 0 0 1 6 7 8 9

	rW
9	1 2 3 4 5 6 7 8 9 0 1 2
8	2 3 4 5 6 7 8 9 0 1 2 3
7	3 4 5 6 7 8 9 0 1 2 3 4
6	4 5 6 7 8 9 0 1 2 3 4 5
5	2 3 4 5 6 7 8 0 0 0 0 0
4	0 0 0 0 0 0 0 9 8 7 6 5
3	0 0 0 0 0 0 0 1 4 5 6 7
2	0 0 0 0 0 0 0 1 5 6 7 8
1	0 0 0 0 0 0 0 1 6 7 8 9

FIGURE 6-7

INSTRUCTION OPERATION

Ynm (m,..., m + 10n − 1)⟶rZ

Transfer the contents of 10n consecutive memory cells, starting with m, to the first 10n cells of rZ.

Y	1	1	8	9	0

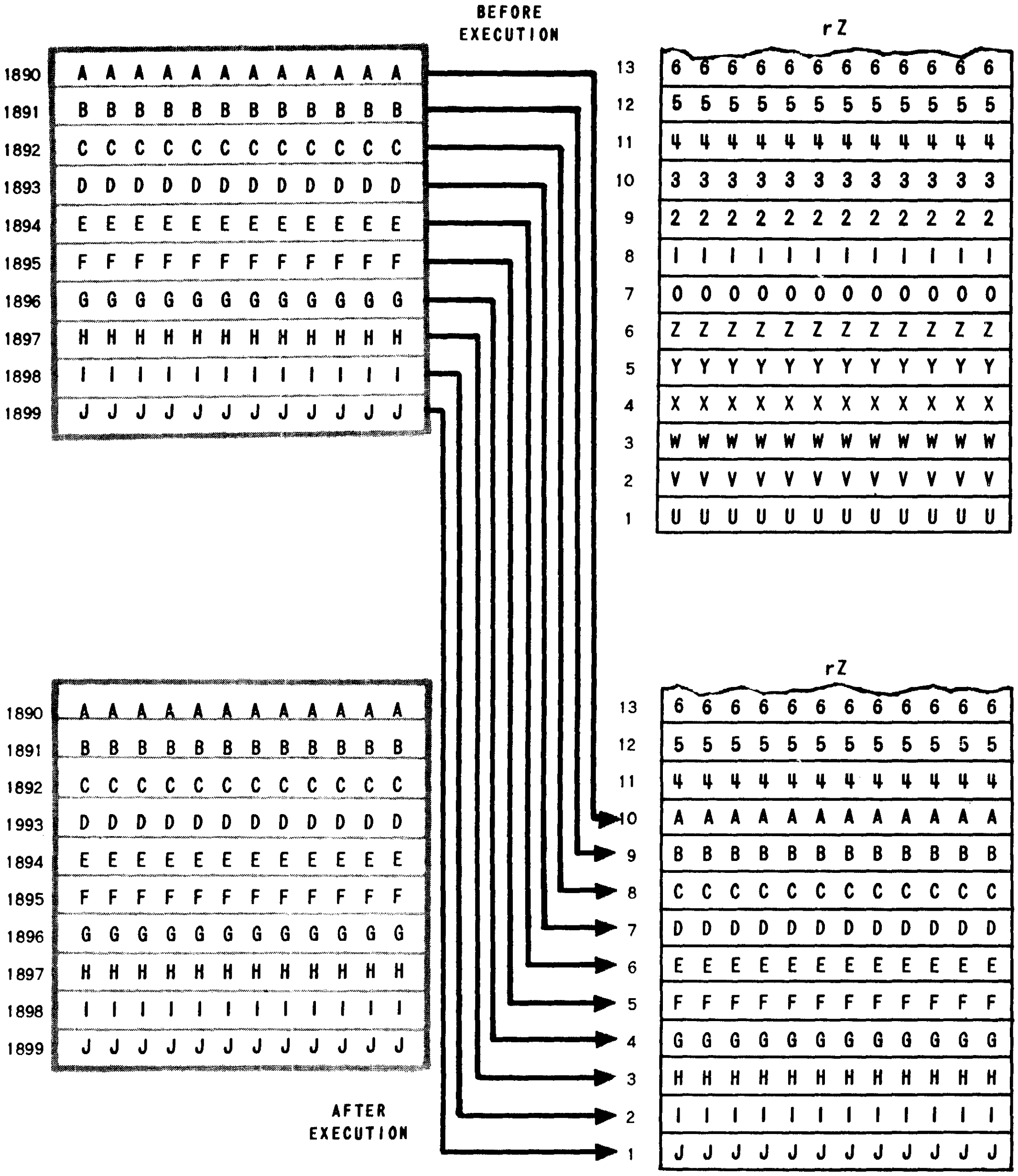
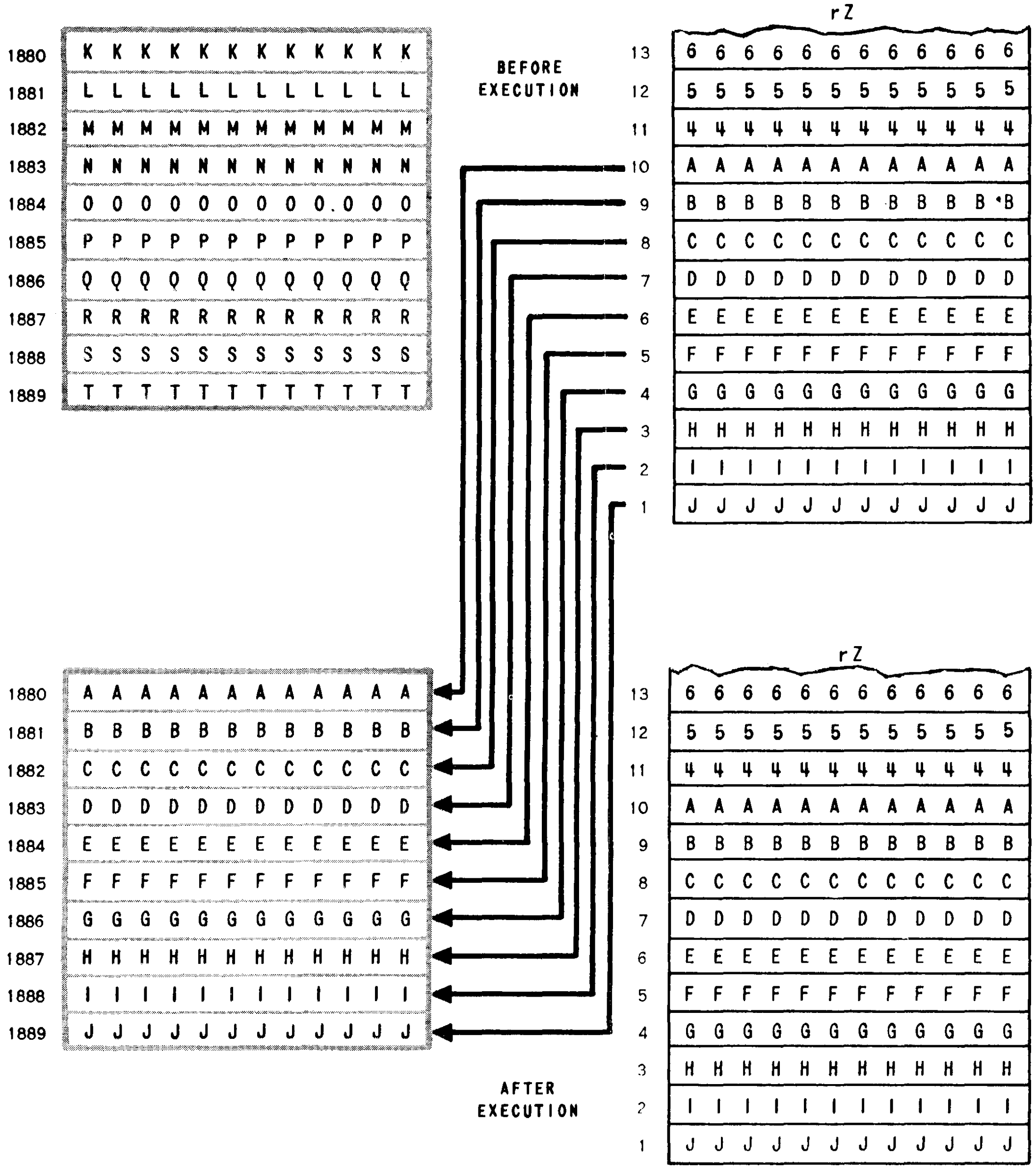

BEFORE EXECUTION
rZ
1890 A A A A A A A A A A A A
1891 B B B B B B B B B B B B
1892 C C C C C C C C C C C C
1893 D D D D D D D D D D D D
1894 E E E E E E E E E E E E
1895 F F F F F F F F F F F F
1896 G G G G G G G G G G G G
1897 H H H H H H H H H H H H
1898 I I I I I I I I I I I I
1899 J J J J J J J J J J J J
13 6 6 6 6 6 6 6 6 6 6 6 6
12 5 5 5 5 5 5 5 5 5 5 5 5
11 4 4 4 4 4 4 4 4 4 4 4 4
10 3 3 3 3 3 3 3 3 3 3 3 3
9 2 2 2 2 2 2 2 2 2 2 2 2
8 I I I I I I I I I I I I
7 0 0 0 0 0 0 0 0 0 0 0 0
6 Z Z Z Z Z Z Z Z Z Z Z Z
5 Y Y Y Y Y Y Y Y Y Y Y Y
4 X X X X X X X X X X X X
3 W W W W W W W W W W W W
2 V V V V V V V V V V V V
1 U U U U U U U U U U U U
rZ
1890 A A A A A A A A A A A A
1891 B B B B B B B B B B B B
1892 C C C C C C C C C C C C
1993 D D D D D D D D D D D D
1894 E E E E E E E E E E E E
1895 F F F F F F F F F F F F
1896 G G G G G G G G G G G G
1897 H H H H H H H H H H H H
1898 I I I I I I I I I I I I
1899 J J J J J J J J J J J J
13 6 6 6 6 6 6 6 6 6 6 6 6
12 5 5 5 5 5 5 5 5 5 5 5 5
11 4 4 4 4 4 4 4 4 4 4 4 4
10 A A A A A A A A A A A A
9 B B B B B B B B B B B B
8 C C C C C C C C C C C C
7 D D D D D D D D D D D D
6 E E E E E E E E E E E E
5 F F F F F F F F F F F F
4 G G G G G G G G G G G G
3 H H H H H H H H H H H H
2 I I I I I I I I I I I I
1 J J J J J J J J J J J J
AFTER EXECUTION

FIGURE 6-8

INSTRUCTION	OPERATION
Znm	(rZ)⟶m,..., m + 10n − 1

Transfer the contents of the first 10n cells of rZ to consecutive memory cells starting with m.

Z	1	1	8	8	0



BEFORE EXECUTION

1880	K K K K K K K K K K K K
1881	L L L L L L L L L L L L
1882	M M M M M M M M M M M M
1883	N N N N N N N N N N N N
1884	0 0 0 0 0 0 0 0 0 0 0 0
1885	P P P P P P P P P P P P
1886	Q Q Q Q Q Q Q Q Q Q Q Q
1887	R R R R R R R R R R R R
1888	S S S S S S S S S S S S
1889	T T T T T T T T T T T T

rZ

13	6 6 6 6 6 6 6 6 6 6 6 6
12	5 5 5 5 5 5 5 5 5 5 5 5
11	4 4 4 4 4 4 4 4 4 4 4 4
10	A A A A A A A A A A A A
9	B B B B B B B B B B B B
8	C C C C C C C C C C C C
7	D D D D D D D D D D D D
6	E E E E E E E E E E E E
5	F F F F F F F F F F F F
4	G G G G G G G G G G G G
3	H H H H H H H H H H H H
2	I I I I I I I I I I I I
1	J J J J J J J J J J J J

AFTER EXECUTION

1880	A A A A A A A A A A A A
1881	B B B B B B B B B B B B
1882	C C C C C C C C C C C C
1883	D D D D D D D D D D D D
1884	E E E E E E E E E E E E
1885	F F F F F F F F F F F F
1886	G G G G G G G G G G G G
1887	H H H H H H H H H H H H
1888	I I I I I I I I I I I I
1889	J J J J J J J J J J J J

rZ

13	6 6 6 6 6 6 6 6 6 6 6 6
12	5 5 5 5 5 5 5 5 5 5 5 5
11	4 4 4 4 4 4 4 4 4 4 4 4
10	A A A A A A A A A A A A
9	B B B B B B B B B B B B
8	C C C C C C C C C C C C
7	D D D D D D D D D D D D
6	E E E E E E E E E E E E
5	F F F F F F F F F F F F
4	G G G G G G G G G G G G
3	H H H H H H H H H H H H
2	I I I I I I I I I I I I
1	J J J J J J J J J J J J

FIGURE 6-9

If the execution of a Ynm instruction is followed by the execution of a Znm instruction, and if the second instruction digit is the same in both instructions, the execution of the Znm instruction will store the words in the same sequence that the execution of the Ynm instruction selected them. The cases where the second instruction digit is not the same in each instruction is explained in a later chapter.

The following coding uses working storage to solve the preceding example.

```
    0000  READ           }
                 DATA    }   read data
(1) 0001  B01880         }
                 C01962  }   J_i^N --> P_i^N
    0002  B01881         |
                 S01882  |
    0003  S01883         |   J_i^C - J_i^L - J_i^M - J_i^O --> P_i^A
                 S01884  |
    0004  C01963         |
                 B00007  }
    0005  L00012         }   i : 12
                 Q00010  }
    ---------------------
    0006  V21962             Output WS-->output area
                 000000
    0007  W21940
                 V51885      Input item-->input WS
    0008  W51880
                 A00013  }   i + 1-->i
    0009  C00007         }
                 U00001
    ---------------------
    0010  WRITE          }
                 DATA    }   write data
    0011  900000             stop
    ---------------------
    0012  W21962         |
                 V51940  |
    0013  000002         |   constants
                 000005  |
```

The item advance coding is in cells 0004-0009. The variable word is in cell 0007. The BOm, LOm and QOm instructions in cells 0004 and 0005 test i against 12. The variable word will be

W21962V51940

immediately after the last item has been processed. The Vnm instruction in cell 0006 and the Wnm instruction in cell 0007 transfer the output item just produced from output working storage to its proper location in the output area. The Vnm instruction in cell 0007 and the Wnm instruction in cell 0008 transfer the next input item from its location in the input area to input working storage. The AOm and COm instructions in cells 0008 and 0009 increase the addresses of the instructions in the variable word. The VOm instruction in cell 0009 transfers control to the processing instructions.

STUDENT EXERCISE

Reading the data stores, in cells 1880-1939, 15 four word inventory items of form

000000NNNNNN
0000000HHHHH$_\wedge$
0000000OOOOO$_\wedge$
0000000RRRRR$_\wedge$

where N - stock number
H - on hand quantity
O - on order quantity
R - minimum required quantity

and in cells 1820-1849, 15 two word sales items of form

0000000NNNNN
0000000QQQQQ$_\wedge$

where N - stock number
Q - sales quantity

The inventory item in cells 1880-1883 and the sales item in cells 1820 and 1821 have the same stock number; the item in cells 1884-1887 and the item in cells 1822 and 1823 have the same stock number; the item in cells 1888-1891 and the item in cells 1824 and 1825 have the same number; and so on. Write the updated inventory items. If the sales quantity for an inventory item reduces the sum of the on hand and on order quantities below the required quantity, print the stock number of the inventory item and the quantity needed to bring the sum back up to the required quantity in form

0000000DDDDD$_\wedge$

FIELD SELECTION INSTRUCTIONS

INSTRUCTION	OPERATION	MNEMONIC
FOm	(m)⟶rF	Fill

Transfer (m) to rF, or <u>fill</u> rF with (m).

INSTRUCTION	OPERATION
GOm	(rF)⟶m

Transfer (rF) to m.

A word may contain more than one field. The shift instructions are one means of separating one field of a word from others. Field selection instructions are used for the same purpose, but are faster and more versatile.

Starting with the "i" and moving up the collation sequence of characters, every other character is called odd. The remaining characters are called even. Recall that the relative magnitude of characters can be determined by reading down the chart, which is figure 2-18.

INSTRUCTION	OPERATION	MNEMONIC
EOm	"odd" characters of (rF) extract (m)⟶rA	Extract

Replace the characters of (rA) that correspond to the odd characters of (rF) with the corresponding characters of (m), or <u>extract</u> (m) into rA.

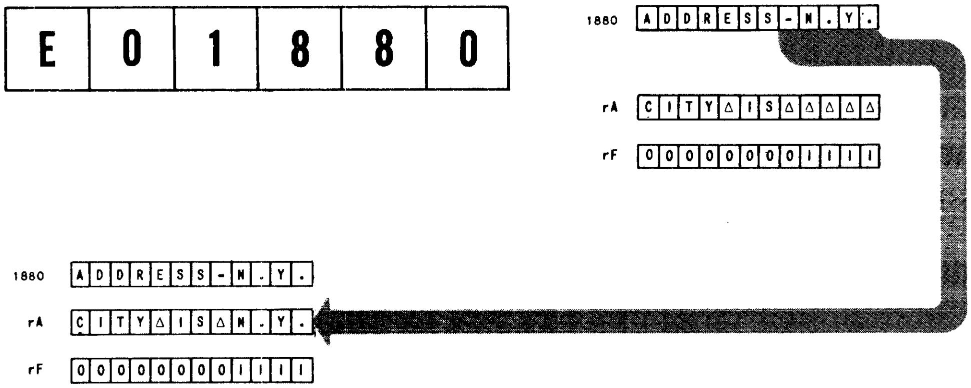
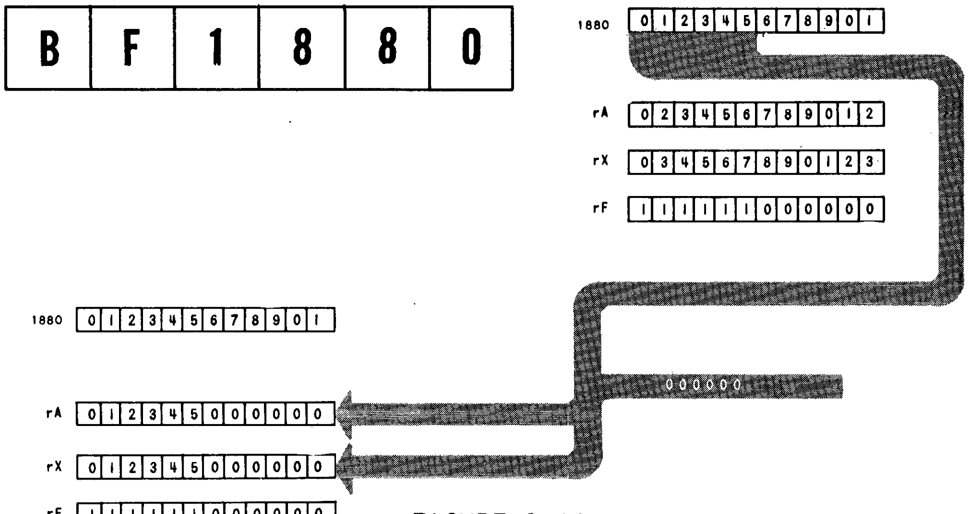

E 0 1 8 8 0
1880 A D D R E S S - N . Y .
rA C I T Y △ I S △ △ △ △ △
rF 0 0 0 0 0 0 0 0 1 1 1 1
1880 A D D R E S S - N . Y .
rA C I T Y △ I S △ N . Y .
rF 0 0 0 0 0 0 0 0 1 1 1 1

FIGURE 6-10

If an F is coded in the second instruction digit of a BOm, AOm, SOm, LOm, POm, MOm, NOm or DOm instruction, the instruction will be executed as usual except that, of the 12 characters of the word being transferred from the memory to the arithmetic unit, only those characters corresponding to the odd characters of (rF) will be transferred; zeros will substituted for the remaining characters.

INSTRUCTION	OPERATION	MNEMONIC
BFm	"odd" characters of(rF) extract (m)→rA, rX	Bring Field Select

Transfer to rA and rX the characters of (m) that correspond to the odd characters of (rF). Transfer zeros to the other digit positions of rA and rX.

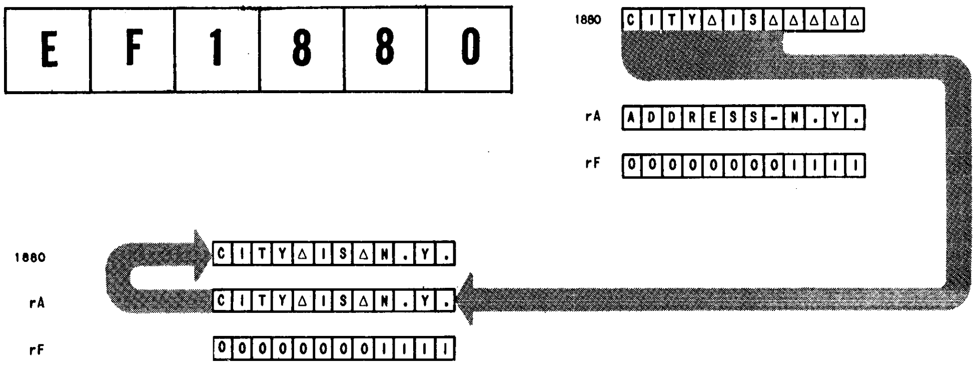


B F 1 8 8 0
1880 0 1 2 3 4 5 6 7 8 9 0 1
rA 0 2 3 4 5 6 7 8 9 0 1 2
rX 0 3 4 5 6 7 8 9 0 1 2 3
rF 1 1 1 1 1 1 0 0 0 0 0 0
1880 0 1 2 3 4 5 6 7 8 9 0 1
000000
rA 0 1 2 3 4 5 0 0 0 0 0 0
rX 0 1 2 3 4 5 0 0 0 0 0 0
rF 1 1 1 1 1 1 0 0 0 0 0 0

FIGURE 6-11

INSTRUCTION	OPERATION
EFm	"even" characters of (rF) extract (m)→rA; (rA)→m

Replace the characters of (rA) that correspond to the even characters of (m). Transfer (rA) to m.



E F 1 8 8 0
1880 C I T Y Δ I S Δ Δ Δ Δ Δ
rA A D D R E S S - N . Y .
rF 0 0 0 0 0 0 0 0 1 1 1 1
1880 C I T Y Δ I S Δ N . Y .
rA C I T Y Δ I S Δ N . Y .
rF 0 0 0 0 0 0 0 0 1 1 1 1

FIGURE 6-12

The field selection instructions facilitate the computer's ability to handle data regardless of how the fields of an item are distributed over words. Consequently, fields can be packed into an item, thus reducing the item size. This reduction results in less tape space per item.

ILLUSTRATIVE EXAMPLE

Reading the data stores, in cells 1880-1939, 20 three word job items of form

```
NNNNNNN00000
00CCCCCLLLLL
     ^    ^
00MMMMMOOOOO
     ^    ^
```

where N - job number
C - contract price
L - labor cost
M - material cost
O - overhead cost

For each job item produce a one word profit item of form

```
NNNNNNNAAAAA
          ^
```

where N - job number
A - profit

Write the profit items.

FLOW CHART

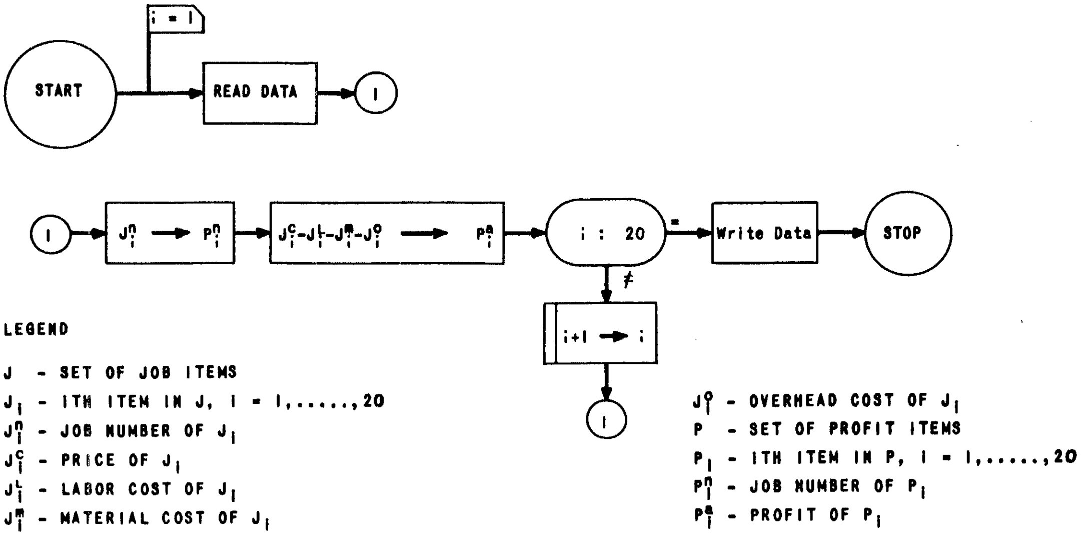

i = 1
START
READ DATA
1
1
$J_i^n \rightarrow P_i^n$
$J_i^c-J_i^L-J_i^m-J_i^o \rightarrow P_i^a$
i : 20
=
Write Data
STOP
≠
$i+1 \rightarrow i$
1
LEGEND
J - SET OF JOB ITEMS
J_i - ITH ITEM IN J, I = 1,.....,20
J_i^n - JOB NUMBER OF J_i
J_i^c - PRICE OF J_i
J_i^L - LABOR COST OF J_i
J_i^m - MATERIAL COST OF J_i
J_i^o - OVERHEAD COST OF J_i
P - SET OF PROFIT ITEMS
P_i - ITH ITEM IN P, I = 1,.....,20
P_i^n - JOB NUMBER OF P_i
P_i^a - PROFIT OF P_i

FIGURE 6-13

CODING

0000	READ		read data
		DATA	
① 0001	B01880		$J_i^N \rightarrow P_i^N$
		C01959	
0002	F00013		001111100000 → rF
		BF1881	00CCCCC00000 → rA, rX
0003	SF1882		$(rA) - J_i^M \rightarrow rA$
		50000	$J_i^C - J_i^M - J_i^L - J_i^O \rightarrow P_i^A$
0004	F00014		
		SF1881	$(rA) - J_i^L \rightarrow rA$
0005	SF1882		$(rA) - J_i^O \rightarrow rA$
		EF1959	$P_j^A \rightarrow m$
0006	B00009		
		L00015	
			i : 20
0007	A00016		
		Q00011	
0008	C00009		i + 1 → i
		B01959	
0009	[C01939		
		V31880]	
0010	W31880		
		U00001	
0011	WRITE		write data
		DATA	
0012	900000		stop
		↪	
0013	001111		
		100000	
0014	↪		
		011111	constants
0015	C01959		
		V31940	
0016	000001		
		000003	

STUDENT EXERCISES

1. Reading the data stores, in cells 1880 and 1881, two one word items of form

$$0\,AAA_{\wedge}0BBB_{\wedge}0CCC_{\wedge}$$

where A, B, and C are numetic quantities. Print the sum of the C fields in form

$$00000000SSSS_\wedge$$

2. Reading the data stores

DATA	FORM	CELL
Quantity A	$000AA_\wedge AA00000$	1880
Quantity B	$000000BBBBBB_\wedge$	1881

Print the sum of the quantities in form

$$000SSSSSSS_\wedge SS$$

3. Reading the data stores in cells 1880-1939, 30 two word census items of form

$$0SS000CCCC00$$
$$0AAA0M0I000G$$

where S - state code
C - city code
A - age
M - marital status code
I - income bracket code
G - sex code

Print the number of single (marital status code S) females (sex code F), age 21 or older, living in Sheboygan (city code 1313), Wisconsin (state code 24), and earning $10,000 or more (income bracket code U).

4. Reading the data stores, in cells 1880-1939, 30 two word inventory items of form

$$NNNNNN0HHHHH_\wedge$$
$$0OOOOO_\wedge RRRRRR_\wedge$$

where N - stock number
H - on hand quantity
O - on order quantity
R - minimum required quantity

and in cells 1820-1849, 30 one word sales items of form

$$NNNNNN0QQQQQ_\wedge$$

where N - stock number
Q - sales quantity

The inventory item in cells 1880 and 1881 and the sales item in cell 1820 have the same stock number, the item in cells 1882 and 1883 and the item in cell 1821 have the same stock number, the item in cells 1884 and 1885 and the item in cell 1822 have the same number, and so on. Write the updated inventory items. If the sales quantity for an inventory item reduces the sum of the on hand and on order quantities below the required quantity, print the stock number of the inventory item and the quantity needed to bring the sum back up to the required quantity in form

NNNNNNΔDDDDD$_\wedge$

where N - stock number
D - quantity needed

5. Design the following items:

1. Inventory item

FIELD	NUMBER OF CHARACTERS
Stock Number	8
Description	24
Unit of measure	1
On-hand amount	5
On-order amount	5
Minimum Reorder Level	5
Unit Price	6
	54

2. Master Employee Item

FIELD	NUMBER OF CHARACTERS
Badge Number	8
Social Security Number	9
Hourly rate of pay	4
Number of exemptions	2
Job description code	2
Year-to-date gross pay	7
Year-to-date FICA tax	6
	[illegible]

3. Transaction Item

FIELD	NUMBER OF CHARACTERS
Key	8
Transaction Code	4
Transaction Information	12
	[illegible]

SUMMARY

An item is a group of characters, usually several words, which completely defines a unit of data.

A field of an item is a group of characters which defines one characteristic of an item.

The notation for field U of the ith item in set D would be D_i^U. $i+1 \rightarrow i$ still indicates the selection of the next item, regardless of the item size.

Working storage is an area used to store one item for input or output advance, or processing.

INSTRUCTIONS

Vnm: n words starting at m → rW
Wnm: n words from rW → memory, starting at m
Ynm: 10n words starting at m → rZ
Znm: 10n words from rZ → memory, starting at m

FOm: (m) → rF
GOm: (rF) → m
EOm: (rF), "odd" characters, extract (m) → rA
EFm: (rF), "even" characters,extract (m) → rA, m

If an F is placed in the second instruction digit of an AOm, BOm, DOm, LOm, MOm, NOm, POm or SOm instruction, the instruction will be executed as defined except that only those characters in (m) which correspond to the odd characters of (rF) will be transferred; zeros will be substituted for the remaining characters.

chapter 7

Subroutines and Variable Connectors

COMMON SUBROUTINES

ILLUSTRATIVE EXAMPLE

Reading the data stores, in cells 1880-1939, 30 two word job items of form

```
0SLLLLLPPPPP
     ^    ^
00MMMMMOOOOO
     ^    ^
```

where S - salesman code and can be

A - if salesman A made the contract
B - if salesman B made the contract

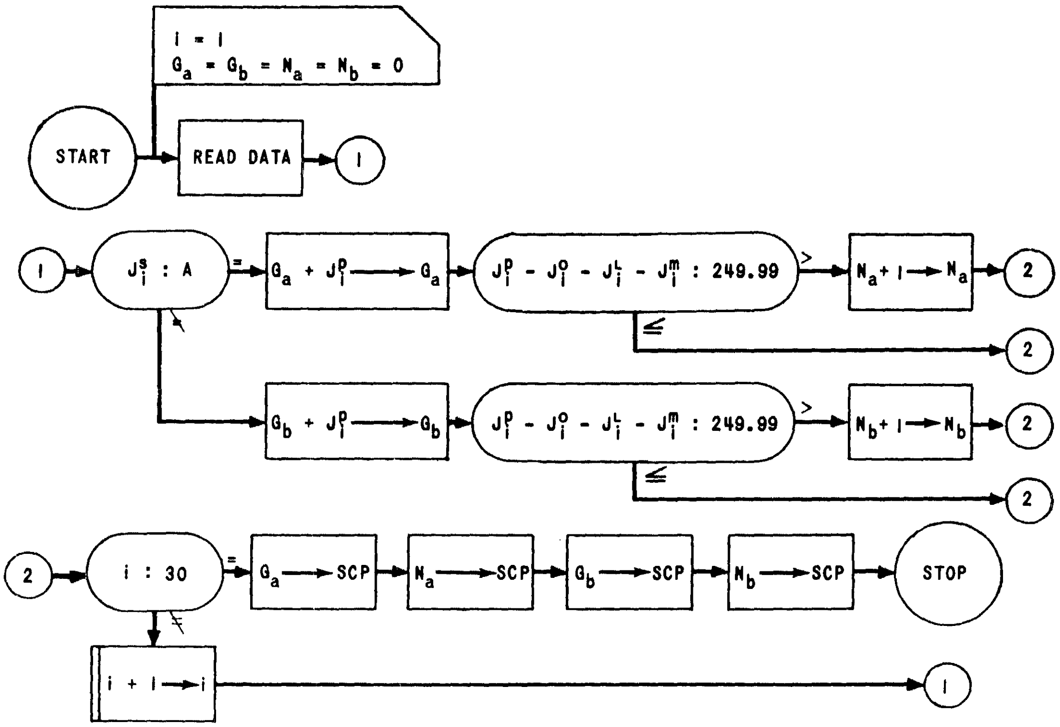
P - contract price
L - labor cost
M - material cost
O - overhead cost

Print

1. the gross sales of salesman A,
2. the number of contracts netting $250 or more made by A,
3. the gross sales of B,

and 4. the number of contracts netting $250 or more made by B.

FLOW CHART



LEGEND

J - SET OF JOB ITEMS

J_i - iTH ITEM IN J, i = 1,...., 30

J_i^s - SALESMAN OF J_i

J_i^p - PRICE OF J_i

J_i^o - OVERHEAD COST OF J_i

J_i^L - LABOR COST OF J_i

J_i^m - MATERIAL COST OF J_i

FIGURE 7-1

CODING

	0000	READ		read data
			DATA	
①	0001	F00031		$J_i^s : A$
			BF1880	
	0002	L00032		
			Q00013	
	0003	B00028		$G_B + J_i^P \rightarrow G_B$
			F00033	
	0004	AF1880		
			C00028	

```
0005   BF1880
                    SF1881
0006   ,50000
                    F00034
0007   SF1880
                    SF1881
0008   L00035
                    T00020
```

$J_i^P - J_i^O - J_i^L - J_i^M$: 249.99

(2)

```
0009   B00011
                    L00036
0010   A00037
                    Q00024
```

i : 30

```
0011 [ V21882
                    W21880 ]
0012   C00011
                    U00001
```

i + 1 → i

```
0013   B00027
                    F00033
0014   AF1880
                    C00027
```

$G_A + J_i^P \longrightarrow G_A$

```
0015   BF1880
                    SF1881
0016   ,50000
                    F00034
0017   SF1880
                    SF1881
0018   L00035
                    T00022
```

$J_i^P - J_i^O - J_i^L - J_i^M$: 249:99

```
0019
                    U00009
```

```
0020   B00029
                    A00038
0021   C00029
                    U00009
```

$N_A + 1 \longrightarrow N_A$

```
0022   B00030
                    A00038
0023   C00030
                    U00009
```

$N_B + 1 \longrightarrow N_B$

```
0024   500027
                    500029
0025   500028
                    500030
0026   900000
```

$G_A \longrightarrow$ SCP
$N_A \longrightarrow$ SCP
$G_B \longrightarrow$ SCP
$N_B \longrightarrow$ SCP
Stop

```
0027  [ ∽                 ]  }
        [         ∽        ]  }   GA
0028  [ ∽                 ]  }
                  ∽        ]  }   GB
0029  [ ∽                 ]  }
                  ∽        ]  }   NA
0030  [ ∽                 ]  }
                  ∽        ]  }   NB
0031    010000                |
                  ∽           |
0032    0A0000                |
                  ∽           |
0033    ∽                     |
                  011111      |
0034    001111                |
                  100000      |
0035    002499                }   constants
                  900000      |
0036    V21942                |
                  W21880      |
0037    000002                |
                  ∽           |
0038    ∽                     |
                  000001      |
```

The coding in cells 0005-0008 is duplicated in cells 0015-0018. This duplication can be eliminated, with the consequence that memory space will be conserved, by means of the programming principle of the common subroutine.

In the flow chart the duplication is shown by the repetition of the relative magnitude test. This test can be made a common subroutine. The subroutine entrance, or starting point, is represented by a triangle with an arrow leaving it; the exit, by a triangle with an arrow entering it. Subroutine symbols are distinguished from each other by letters, the letter used for a particular subroutine usually being a mnemonic for the operation done by the subroutine. In the following the letter P is used for "profit".

Whenever, on a logical line of flow, it is desired that a subroutine be executed two concentric circles containing the letter of the subroutine are drawn. This symbol means that, once the subroutine exit is reached, the logical line of flow continues from the point where the subroutine was entered.

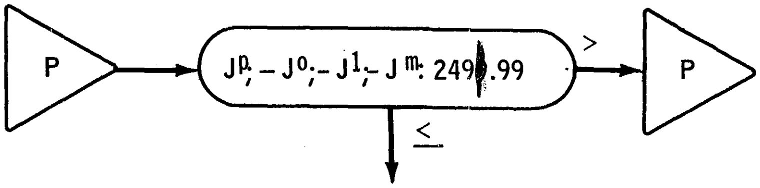
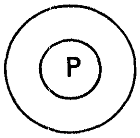


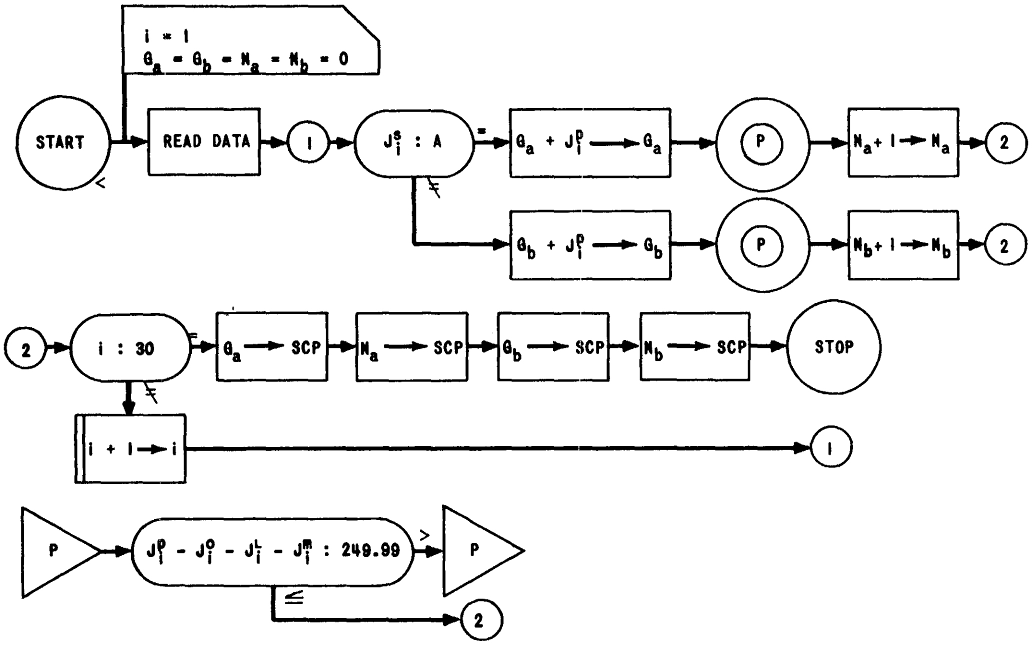
FIGURE 7-2

For example, in the following flow chart (Figure 7-4) after the operation, $G_A + J_i^p \longrightarrow G_A$, the subroutine symbol

FIGURE 7-3



means "execute subroutine P, and when the subroutine exit is reached, continue with the operation, $N_A + 1 \longrightarrow N_A$".



LEGEND

J - SET OF JOB ITEMS

J_i - ITH ITEM IN J, i = 1,...., 30

J_i^s - SALESMAN OF J_i

J_i^p - PRICE OF J_i

J_i^o - OVERHEAD COST OF J_i

J_i^L - LABOR COST OF J_i

J_i^m - MATERIAL COST OF J_i

FIGURE 7-4

In coding from a flow chart containing common subroutines, everytime the logical line of flow encounters a subroutine symbol, it is necessary to code a UOm instruction to transfer control to the common subroutine entrance. When the common subroutine exit is reached, another UOm instruction is needed to transfer control back to the point in the coding from which control was originally transferred. But since the common subroutine may be entered from more than one point in the coding, the address portion of the UOm instruction at the common subroutine exit cannot be fixed, but must vary according to the point in the coding from which the common subroutine was entered. For example, if a common subroutine can be entered by means of a UOm instruction in cell 0005 and also by means of a UOm instruction in cell 0010, the UOm instruction at the common subroutine exit must at times be U00006, and at other times be U00011. In this situation the ROm instruction is useful.

INSTRUCTION	OPERATION	MNEMONIC
ROm	000000U0(CC)⟶m	Record

Store a word consisting of six zeros, a U, a zero and the four least significant digits of (CC) in m, or record 000000U0(CC) in m.

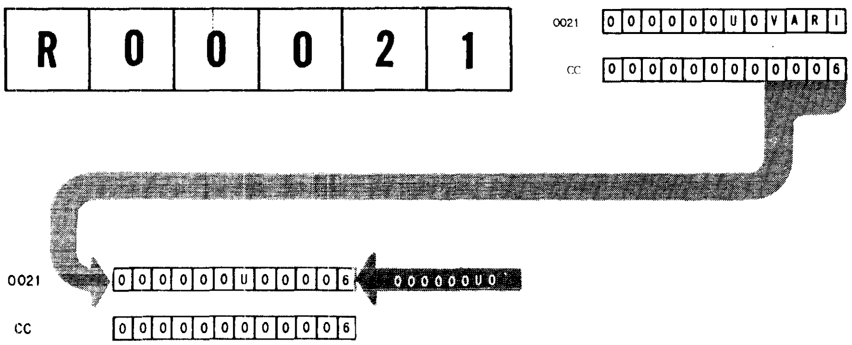

R 0 0 0 2 1
0021 0 0 0 0 0 0 U 0 V A R I
CC 0 0 0 0 0 0 0 0 0 0 0 6
0021 0 0 0 0 0 0 U 0 0 0 0 6
000000U0
CC 0 0 0 0 0 0 0 0 0 0 0 6

FIGURE 7-5

Consider the following. (BTO has just been completed)

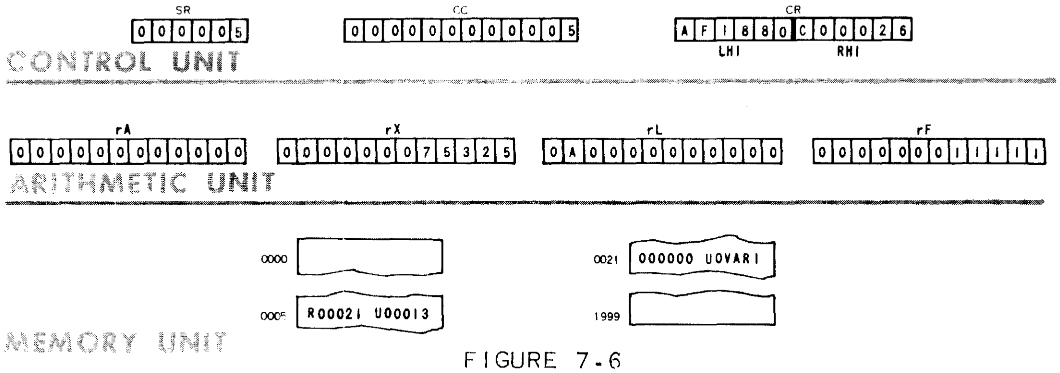


FIGURE 7-6

On β Time On

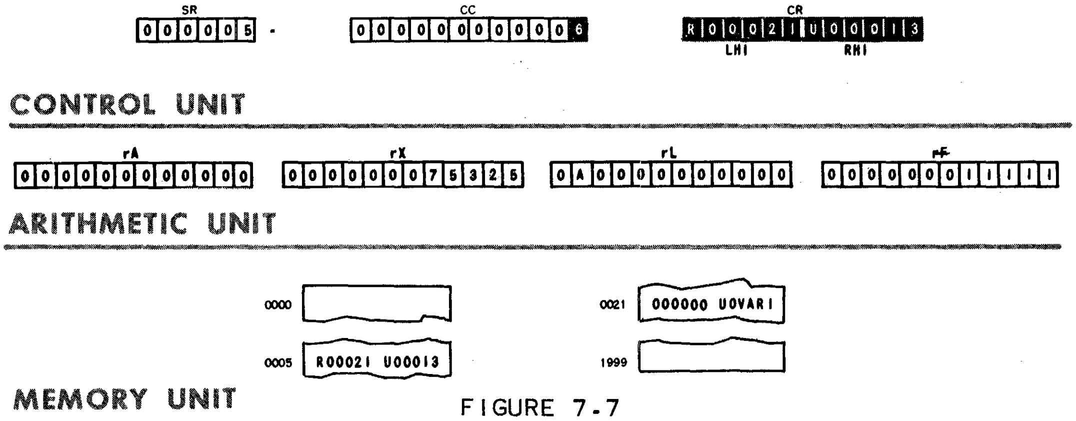


FIGURE 7-7

On γ Time

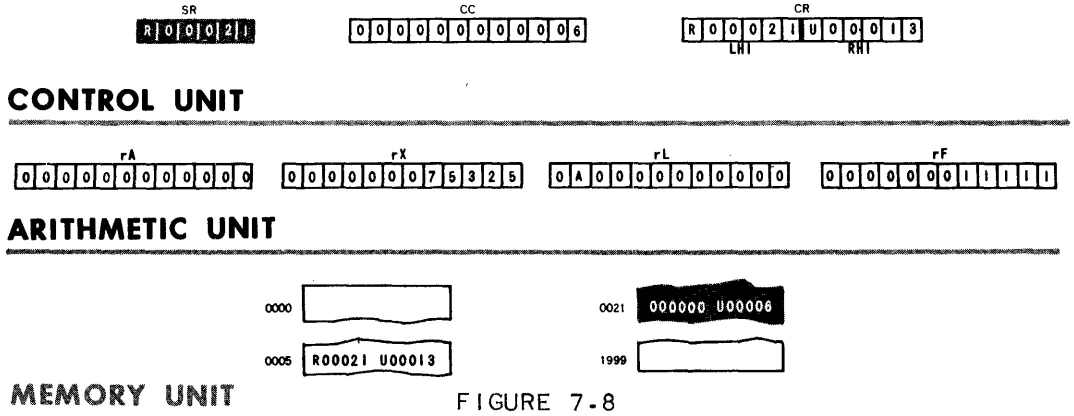


FIGURE 7-8

On δ Time

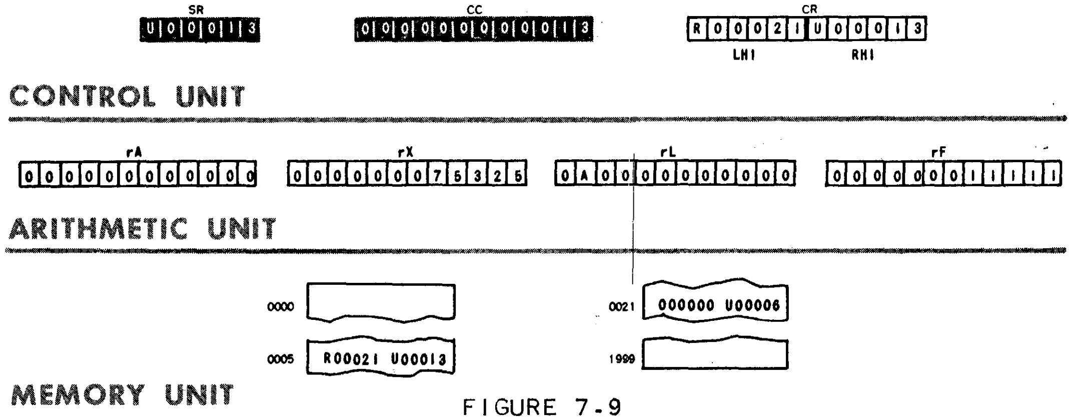


FIGURE 7-9

If cell 0013 were the entrance of a common subroutine; and cell 0021, the exit; and if the common subroutine were to be entered from cell 0005; the execution of the

UOm instruction transfers control to the common subroutine. The ROm instruction executed on γ Time guarantees that, when the common subroutine exit is reached, the instruction pair

```
000000
 U00006
```

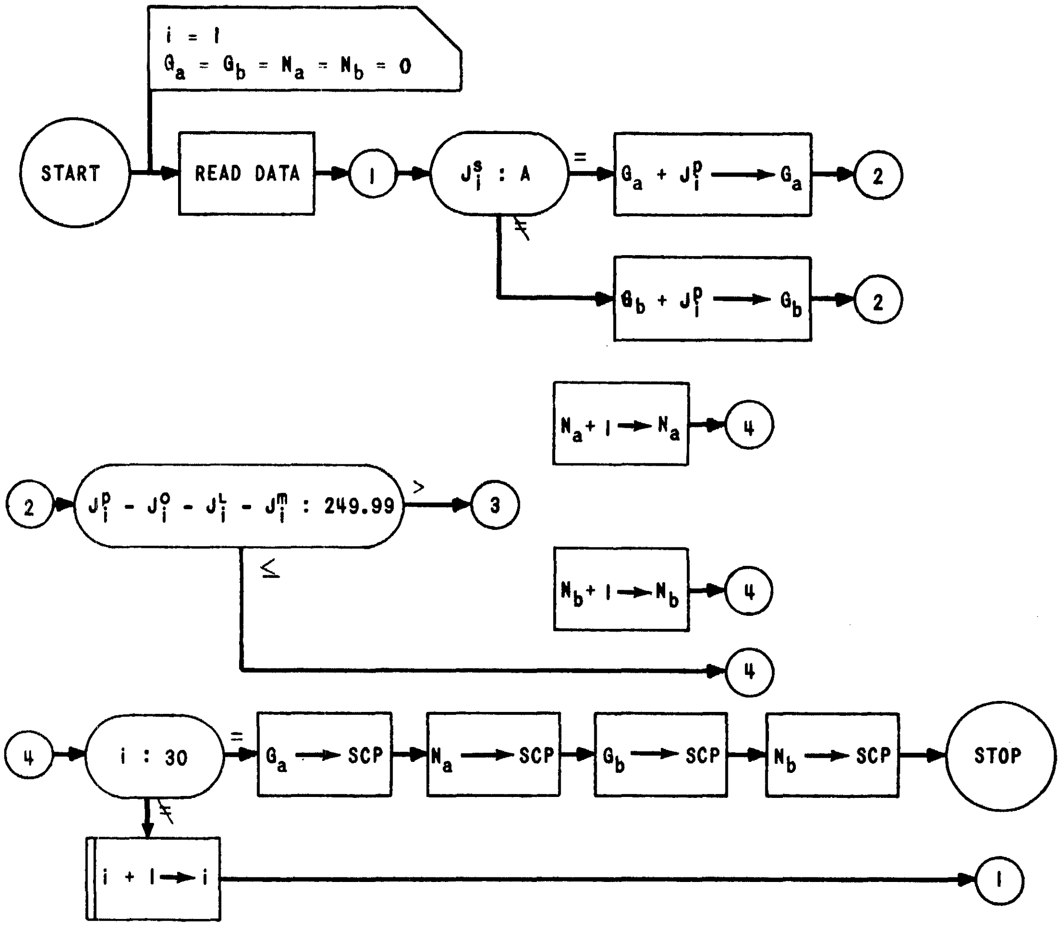
will be executed, transferring control to cell 0006, to continue the processing begun before transferring to the subroutine.

```
     0000   READ             }   read data
                   DATA      }
(1)  0001   F00029           }
                   BF1880    }   J_i^S : A
     0002   L00030           }
                   Q0008     }
     ----------------------
     0003   B00026           }
                   F00031    }   G_B + J_i^P --> G_B
     0004   AF1880           }
                   C00026    }
     0005   R00021           }   (P)          000000U0(CC)--> m
                   U00013    }
     ----------------------
     0006   B00028           }
                   A00032    }   N_B + 1 --> N_B
     0007   C00028           }
                   U00017    }
     ----------------------
     0008   B00025           }
                   F00031    }   G_A + J_i^P --> G_A
     0009   AF1880           }
                   C00025    }
     0010   R00021           }   (P)          000000U0(CC)--> m
                   U00013    }
     ----------------------
     0011   B00027           }   N_A + 1 --> N_A
                   A00032    }
     0012   C00027
                   U00017
     ----------------------
P >  0013   BF1880           }
                   SF1881    }
     0014   , 50000          }
                   F00033    }   J_i^P - J_i^O - J_i^L - J_i^M : 249.99
     0015   SF1880           }
                   SF1881    }
     0016   L00034           }
                   T00021    }
     ----------------------
```

```
②  0017    B00019                       i : 30
                      L00035
    0018    A00036
                      Q00022
    0019   [V21882
                      W21880]
    0020    C00019                       i + 1 → i
                      U00001
    0021   [⁓
                      UOVARI]            ▷P
    0022    500025                       GA → SCP
                      500027             NA → SCP
    0023    500026                       GB → SCP
                      500028             NB → SCP
    0024    900000                       stop
                      ⁓
    0025   [⁓                            GA
                      ⁓]
    0026   [⁓                            GB
                      ⁓]
    0027   [⁓                            NA
                      ⁓]
    0028   [⁓                            NB
                      ⁓]
    0029    010000
                      ⁓
    0030    0A0000
                      ⁓
    0031    ⁓
                      011111
    0032    ⁓
                      000001
    0033    001111                       constants
                      100000
    0034    002499
                      900000
    0035    V21942
                      W21880
    0036    000002
                      ⁓
```

VARIABLE CONNECTORS

The example can be flow charted in another way. (The notation from ③ to ④ in figure 7-10 is incomplete).


i = 1
$G_a = G_b = N_a = N_b = 0$
START
READ DATA
1
$J_i^s : A$
=
≠
$G_a + J_i^p \longrightarrow G_a$
$G_b + J_i^p \longrightarrow G_b$
2
2
$N_a + 1 \rightarrow N_a$
4
2
$J_i^p - J_i^o - J_i^L - J_i^m : 249.99$
>
3
≤
$N_b + 1 \rightarrow N_b$
4
4
4
i : 30
=
≠
$G_a \longrightarrow SCP$
$N_a \longrightarrow SCP$
$G_b \longrightarrow SCP$
$N_b \longrightarrow SCP$
STOP
$i + 1 \rightarrow i$
1

LEGEND

J - A SET OF JOB ITEMS

J_i - THE ITH ITEM IN J, i = 1,...., 30

J_i^s - THE SALESMEN OF J_i

J_i^p - THE PRICE OF J_i

J_i^o - THE OVERHEAD COST OF J_i

J_i^L - THE LABOR COST OF J_i

J_i^m - THE MATERIAL COST OF J_i

FIGURE 7-10

This flow chart has a point of indetermination at connector three. In some cases the logical line of flow is to the operation, $N_A + 1 \longrightarrow N_A$; in other cases, to the operation, $N_B + 1 \longrightarrow N_B$. Thus, connector three must be variable. That is, connector three must act as a switch, sometimes switching the logical line of flow to one operation; sometimes, to the other - just as a railroad switch sometimes switches a train to one track; sometimes, to another. A variable connector is actually represented on a flow chart as a switch, with poles and a terminal. The terminal is a connector with a subscript "v" to the number. The poles are connectors with consecutive alphabetic subscripts to the number.

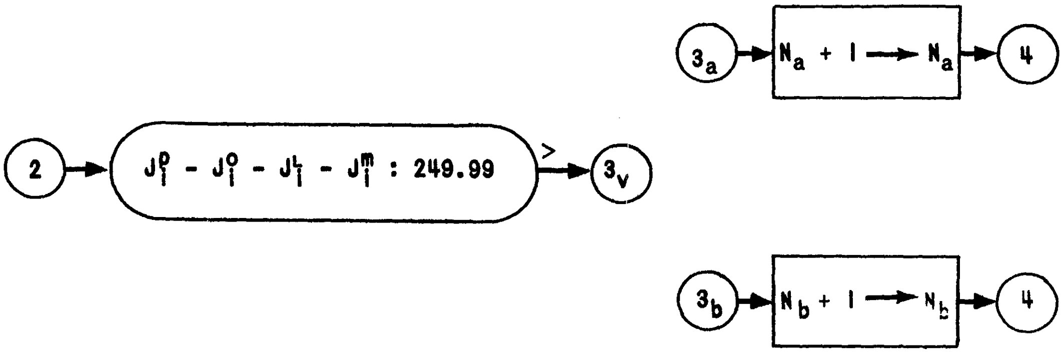

3_a
$N_a + 1 \rightarrow N_a$
4
2
$J_1^p - J_1^o - J_1^L - J_1^m : 249.99$
>
3_v
3_b
$N_b + 1 \rightarrow N_b$
4

FIGURE 7-11

For clarity, the terminal of the variable connector should be symmetrical with the poles.

For a variable connector to operate correctly, it must be set, just as a switch is set. The setting of a variable connector is represented on a flow chart as a square, called a set box, containing a period and the pole of the connector to be set. For example, the set box

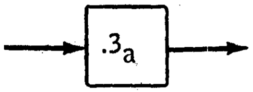
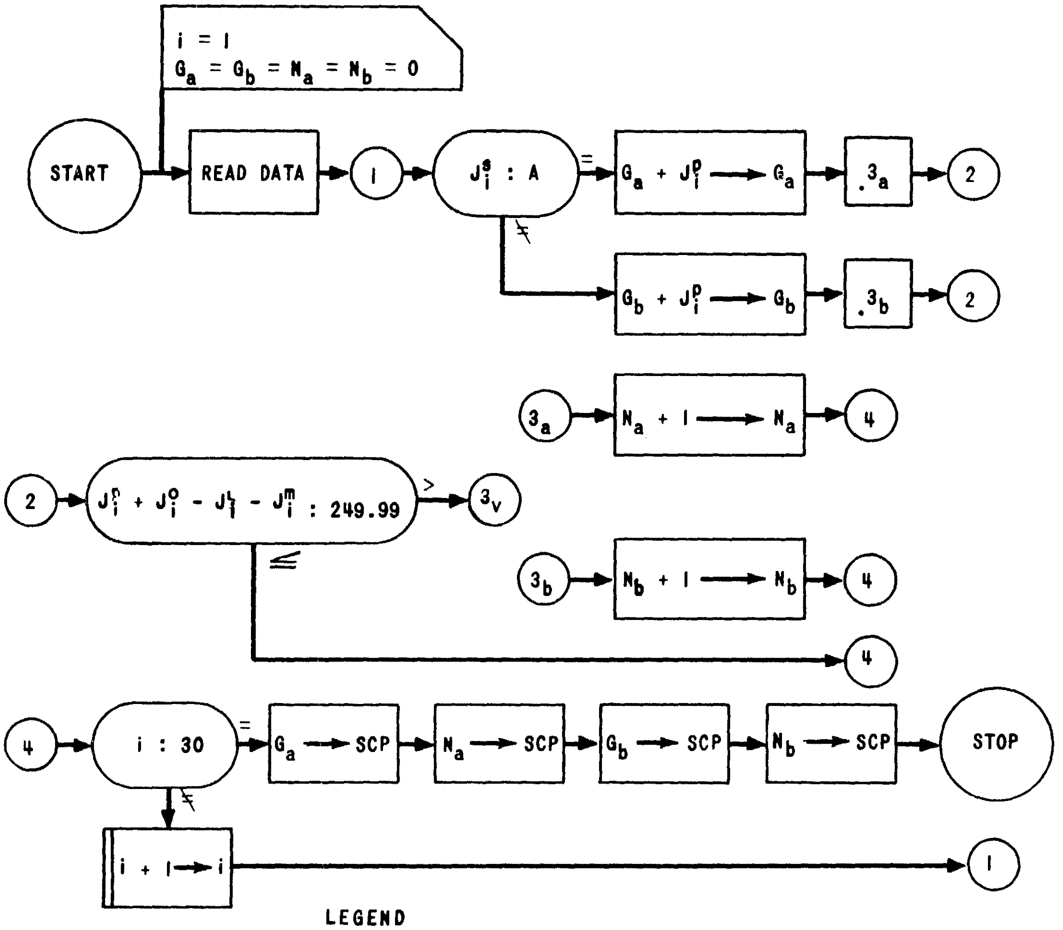


FIGURE 7-12

means that, when the logical line of flow reaches the terminal of variable connector three, it will be switched to pole a. Just as the controls that operate a railroad switch may be separated from the switch by an intervening distance, the set box that sets a variable connector may be, and usually is, separated from the variable connector by intervening operations. In the following flow chart, the test for relative magnitude intervenes between the set boxes for variable connector three and the variable connector itself.



i = 1
$G_a = G_b = N_a = N_b = 0$
START
READ DATA
1
$J_i^s : A$
=
≠
$G_a + J_i^p \longrightarrow G_a$
$.3_a$
2
$G_b + J_i^p \longrightarrow G_b$
$.3_b$
2
3_a
$N_a + 1 \longrightarrow N_a$
4
2
$J_i^p + J_i^o - J_i^L - J_i^m : 249.99$
>
3_v
≤
3_b
$N_b + 1 \longrightarrow N_b$
4
4
4
i : 30
=
≠
$G_a \longrightarrow SCP$
$N_a \longrightarrow SCP$
$G_b \longrightarrow SCP$
$N_b \longrightarrow SCP$
STOP
$i + 1 \longrightarrow i$
1

LEGEND

J - SET OF JOB ITEMS

J_i - ITH ITEM IN J, i = 1,...., 30

J_i^s - SALESMAN OF J_i

J_i^p - PRICE OF J_i

J_i^o - OVERHEAD COST OF J_i

J_i^L - LABOR COST OF J_i

J_i^m - MATERIAL COST OF J_i

FIGURE 7-13

This flow chart is logically equivalent to the flow chart in figure 7-4 and can be coded in the same way. The difference between the two is that one uses the programming principle of the common subroutine; the other, the principle of the variable connector. The programming principle of the variable connector is more general than that of the common subroutine and is used many times when there is no common subroutine. For example, it often occurs in a problem that for a certain number of items to be processed a given operation must be performed, but for the processing of the remainder of the items the operation is not necessary.

The operation can be removed from the processing coding by means of a variable connector.

Variable connectors can be set by means other than the use of the ROm instruction. Setting the variable connectors with BOm COm instruction pairs, the coding for the example might be as follows.

```
     0000    READ                   read data
                       DATA
①    0001    F00029
                       BF1880       $J_i^S : A$
     0002    L00030
                       Q00014
     ---------------------
     0003    B00026
                       F00031       $G_B + J_i^P \rightarrow G_B$
     0004    AF1880
                       C00026
     0005    B00032                 .3b
                       C00009
②    0006    BF1880
                       SF1881
     0007    , 50000
                       F00033       $J_i^P - J_i^O - J_i^L - J_i^M$   :249.99
     0008    SF1880
                       SF1881
③    0009   [L00034
                       TOVARI]
     ---------------------
④    0010    B00012
                       L00035       i : 30
     0011    A00037
                       Q00022
     ---------------------
     0012   [V21882
                       W21880]
     0013    C00012                 i + 1 → i
                       U00001
     ---------------------
     0014    B00025
                       F00031       $G_A + J_i^P \rightarrow G_A$
     0015    AF1880
                       C00025
     0016    B00036                 .3a
                       C00009
     0017
                       U00006
     ---------------------
```

```
(3a)  0018   B00027               ⎫
                        A00038    ⎬   NA + 1 → NA
      0019   C00027               ⎭
                        U00010
      ______________________________
(3b)  0020   B00028               ⎫
                        A00038    ⎬   NB + 1 → NB
      0021   C00028               ⎭
                        U00010
      ______________________________
      0022   500025                   GA → SCP
                        500026        NA → SCP
      0023   500027                   GB → SCP
                        500028        NA → SCP
      0024   900000                   stop
      ______________________________
                        ~~~~~
      0025   [~~~~~               }
                        ~~~~~]    }   GA
      0026   [~~~~~               }
                        ~~~~~]    }   GB
      0027   [~~~~~               }
                        ~~~~~]    }   NA
      0028   [~~~~~               }
                        ~~~~~]    }   NB
      0029   010000               ⎫
                        ~~~~~     ⎪
      0030   0A0000               ⎪
                        ~~~~~     ⎪
      0031   ~~~~~                ⎪
                        011111    ⎪
      0032   L00034               ⎪
                        T00020    ⎪
      0033   001111               ⎪
                        100000    ⎬   constants
      0034   002499               ⎪
                        900000    ⎪
      0035   V21942               ⎪
                        W21880    ⎪
      0036   L00034               ⎪
                        T00018    ⎪
      0037   000002               ⎪
                        ~~~~~     ⎪
      0038   ~~~~~                ⎪
                        000001    ⎭
```

In this coding variable connector three is embodied in the address part of the TOm instruction in cell 0009. This address part varies between 0018 and 0020, depending on whether control is to be switched to pole a or b. The variable connector is set to pole a by the BOm COm instruction pair in cell 0016, to pole b by the pair in cell 0005.

In some flow charts using variable connectors it occurs that initially a variable connector should be set to some given state. This fact is indicated by showing the notation for the setting of the variable connector, not in a set box, but in the assertion flag.

STUDENT EXERCISE

Reading the data stores:

1. six ten word A items in cells 1880-1939
2. six ten word B items in cells 1820-1879

Each A item has for its first word a key, and the items are in ascending order by key. Similar remarks hold for the B items. Create a set of 12 items, consisting of the six A items and the six B items, which is in ascending order by key. (Such an operation is called a "merge"). Write the merged items.

SUBROUTINES

The coding that, when executed, performs a large operation is called a routine. The coding that performs a payroll operation could be called a payroll routine.

The coding that, when executed, does a suboperation of a routine is called a subroutine. A payroll routine might consist of the following subroutines.

1. Determination of gross pay.
2. Determination of medical pay.
3. Determination of withholding tax.
4. Determination of FICA tax.
5. Determination of group insurance contribution.
6. Determination of union dues.
7. Determination of net pay.
8. Item advance.

Using the concept of the subroutine, a routine can be organized into

1. a set of subroutines,

and 2. a framework, or main chain, which specifies the order in which the subroutines are to be executed and performs minor processing.

For example, the payroll routine might be flow charted as follows.

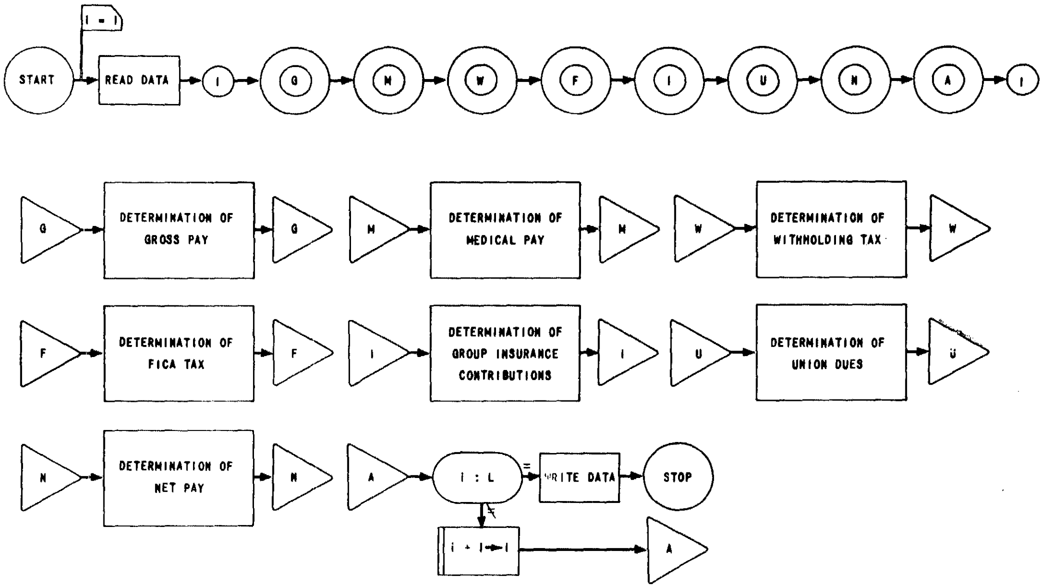

I = 1
START
READ DATA
I
G
M
W
F
I
U
N
A
I
G
DETERMINATION OF GROSS PAY
G
M
DETERMINATION OF MEDICAL PAY
M
W
DETERMINATION OF WITHHOLDING TAX
W
F
DETERMINATION OF FICA TAX
F
I
DETERMINATION OF GROUP INSURANCE CONTRIBUTIONS
I
U
DETERMINATION OF UNION DUES
U
N
DETERMINATION OF NET PAY
N
A
I : L
=
WRITE DATA
STOP
≠
i + 1→I
A

FIGURE 7-14

The subroutine concept allows the programmer to flow chart first in terms of subroutines. He can then flow chart each subroutine as an essentially distinct entity. The subroutine concept not only saves memory space when used with respect to the common subroutine, but also simplifies both the flow charting and coding of a complex routine. Therefore, all of the following problems will be flow charted and coded in subroutine form. Generally each subroutine performs one operation and may be categorized as follows:

1. Starting subroutine - initial operations
2. Input subroutines
3. Processing subroutines
4. Output subroutines
5. Ending subroutine

In the illustrative and student exercises in this manual initial, processing, and ending operations, because they are short, may be coded in the main chain of the program. Whenever these operations are lengthy or detailed, however, they should be treated as distinct subroutines.

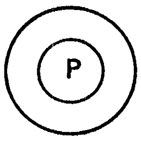
SUMMARY

A routine is the coding for a complete operation.

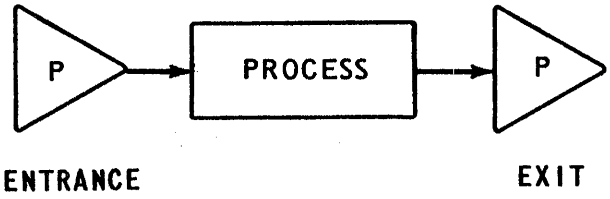
A subroutine performs one suboperation of a routine, and is a distinct sequence of instructions.

A common subroutine is one which is common to several parts of the routine. Its exit must, therefore, be variable.

Because of the advantages of coding subroutines they are used even where not common to several parts of the routine.



Execute subroutine P



Subroutine P

FIGURE 7-15

A variable connector is a connector which allows for alternate paths of processing. Its use is usually to perform:

a. one operation for a time and then a different operation
b. one of a series of possible operations depending on the conditions imposed

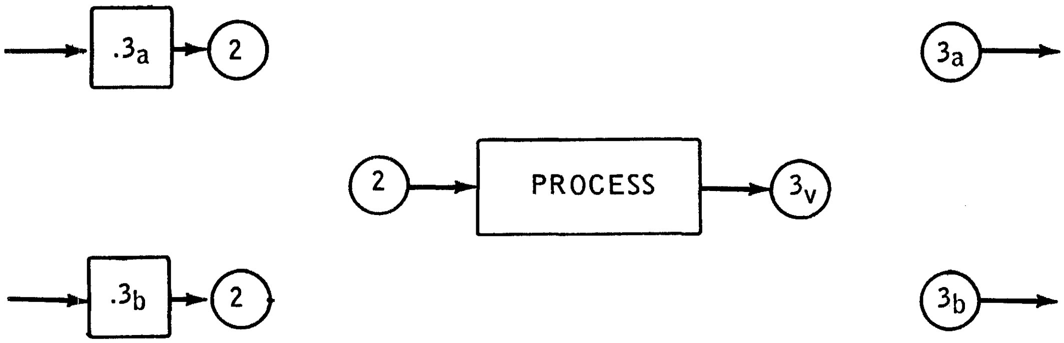


FIGURE 7-16

INSTRUCTION	OPERATION	MNEMONIC
ROm	000000U0(CC)→m	Record

Replace (m) with a skip instruction and a U instruction to the address specified by (CC).

R0XXXXU0YYYY pair of instructions can always be coded for

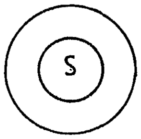


FIGURE 7-17

where XXXX is the address of the exit entrance and YYYY the address of the entrance ~~exit~~ of the subroutine.

chapter 8

Detailed Description of Instructions

TRANSFER OF CONTROL INSTRUCTIONS (for review, see pages 63, 67)

It has been stated that for the proper execution of the instructions, UOm, QOm and TOm, the address part of the instruction must be the four least significant digits of the word in which the instruction appears. Up to this point this requirement has been met by always coding a transfer of control instruction as a RHI. In certain situations it is possible and advantageous to code a transfer of control instruction as a LHI, and the above requirement can still be met.

Suppose that one processing path is to be taken if the contents of cell 1820 are greater than or equal to the contents of cell 1880, and another is to be taken if the contents of cell 1820 are less than the contents of cell 1880. The coding might be

```
0010   B01820
                   L01880
0011
                   T00020
__________________________
0012
                   Q00020
__________________________
```

In this coding, the LHI in cells 0011 and 0012 are wasted, since they are skips. It would be more efficient if the QOm instruction were the LHI in cell 0011. This situation is possible, since the address part of the QOm will still be the four least significant digits of the word in which it appears.

```
0010   B01820
                 L01880
0011   Q00000
                 T00020
```

If the contents of cells 1820 and 1880 are unequal, the QOm instruction will be interpreted as a skip, and the coding takes an already familiar form. If the contents of the cells are equal, the following occurs.

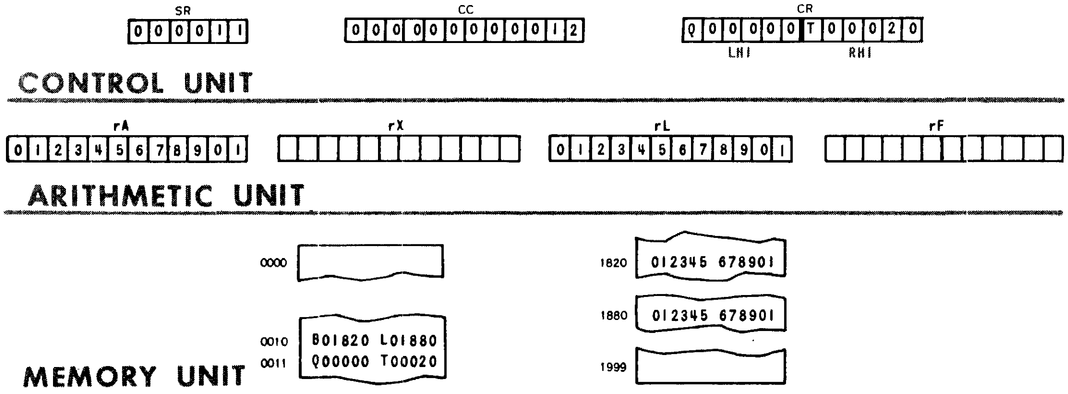


FIGURE 8-1

Assume that the computer has just completed β Time. (CC) specify that the next instruction pair is in cell 0012. On γ Time Q00000 is transferred to SR and executed. Since (rA) are equal to (rL), the execution transfers the four least significant digits of (CR) to CC.

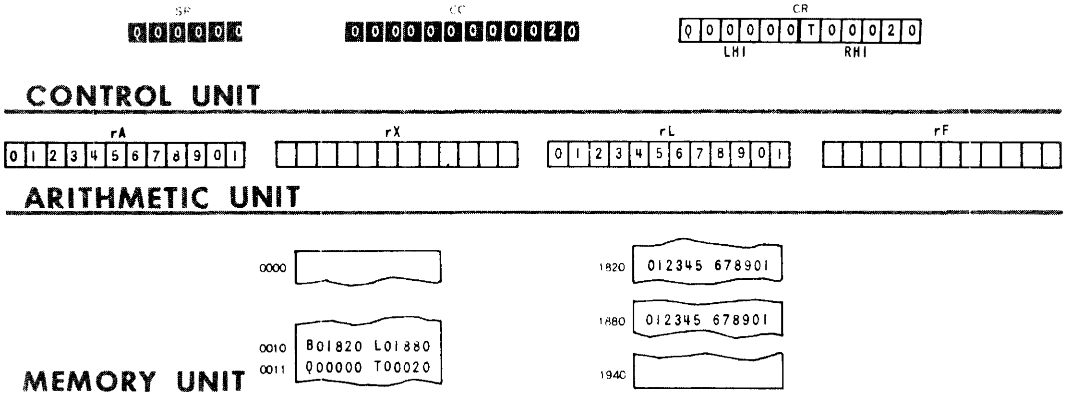


FIGURE 8-2

(CC) now specify that the next instruction pair is in cell 0020, where the coding for the condition of equality begins. On δ Time T00020 is transferred to SR and executed. Since (rA) are not greater than (rL), T00020 is interpreted as a skip.

SHIFT INSTRUCTIONS (for review, see pages 102, 103).

Any character other than a 0-9 in the second instruction digit of a shift instruction, Onm, - nm, ;nm or .nm, causes the computer to stall and light a neon on the Supervisory Control Panel to indicate that an instruction has been improperly coded. A zero in the second instruction digit of a Onm instruction transforms the instruction into a skip instruction. A zero in the second instruction digit of any other shift instruction causes the computer to stall and light a neon indicating that it has stalled.

MULTIWORD TRANSFER INSTRUCTIONS (for review, see pages 115-121).

The question might arise as to what happens if the second instruction digit of a Vnm instruction is different from the second instruction digit of the Wnm instruction that follows it.

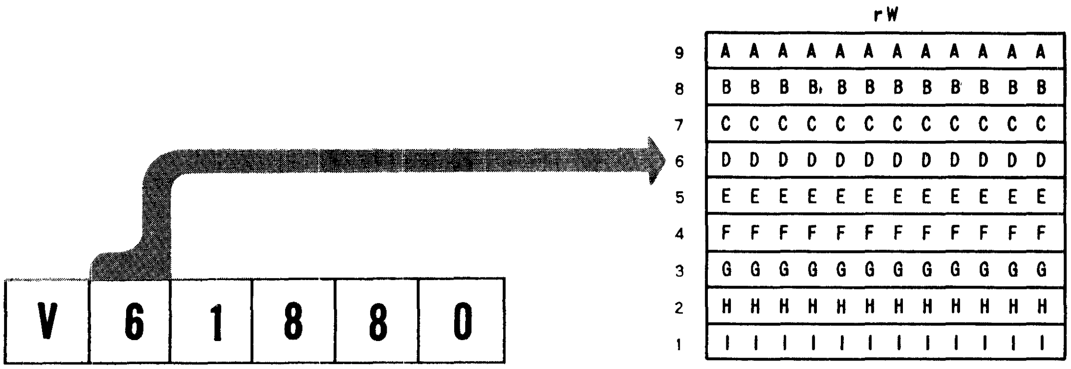
The cells of rW can be thought of as being numbered 1-9 as shown below.

rW

9	A	A	A	A	A	A	A	A	A	A	A	A
8	B	B	B	B	B	B	B	B	B	B	B	B
7	C	C	C	C	C	C	C	C	C	C	C	C
6	D	D	D	D	D	D	D	D	D	D	D	D
5	E	E	E	E	E	E	E	E	E	E	E	E
4	F	F	F	F	F	F	F	F	F	F	F	F
3	G	G	G	G	G	G	G	G	G	G	G	G
2	H	H	H	H	H	H	H	H	H	H	H	H
1	I	I	I	I	I	I	I	I	I	I	I	I

FIGURE 8-3

When a Vnm or Wnm instruction is executed a "pointer" is set to the cell indicated by n.



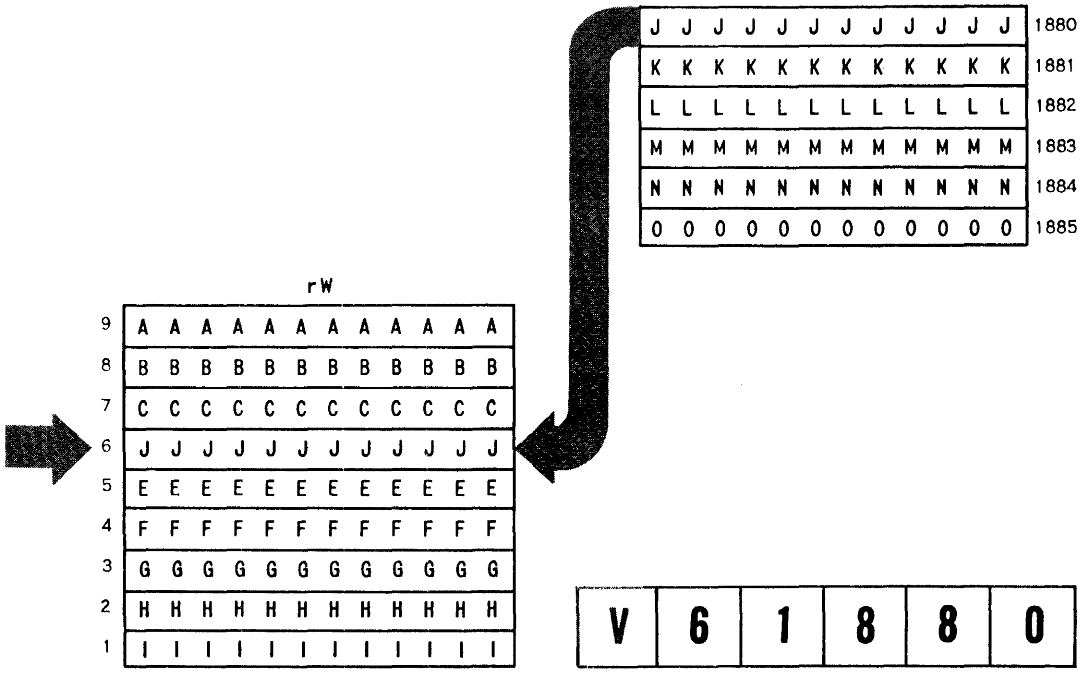
V	6	1	8	8	0

rW

9	A	A	A	A	A	A	A	A	A	A	A	A
8	B	B	B	B	B	B	B	B	B	B	B	B
7	C	C	C	C	C	C	C	C	C	C	C	C
6	D	D	D	D	D	D	D	D	D	D	D	D
5	E	E	E	E	E	E	E	E	E	E	E	E
4	F	F	F	F	F	F	F	F	F	F	F	F
3	G	G	G	G	G	G	G	G	G	G	G	G
2	H	H	H	H	H	H	H	H	H	H	H	H
1	I	I	I	I	I	I	I	I	I	I	I	I

FIGURE 8.4

The first word to be transferred to or from rW is stored in, or transferred from, the cell indicated by the pointer.



J	J	J	J	J	J	J	J	J	J	J	J	1880
K	K	K	K	K	K	K	K	K	K	K	K	1881
L	L	L	L	L	L	L	L	L	L	L	L	1882
M	M	M	M	M	M	M	M	M	M	M	M	1883
N	N	N	N	N	N	N	N	N	N	N	N	1884
O	O	O	O	O	O	O	O	O	O	O	O	1885

rW

9	A	A	A	A	A	A	A	A	A	A	A	A
8	B	B	B	B	B	B	B	B	B	B	B	B
7	C	C	C	C	C	C	C	C	C	C	C	C
6	J	J	J	J	J	J	J	J	J	J	J	J
5	E	E	E	E	E	E	E	E	E	E	E	E
4	F	F	F	F	F	F	F	F	F	F	F	F
3	G	G	G	G	G	G	G	G	G	G	G	G
2	H	H	H	H	H	H	H	H	H	H	H	H
1	I	I	I	I	I	I	I	I	I	I	I	I

V	6	1	8	8	0

FIGURE 8.5

The pointer then moves to the next lower numbered cell, and the second word to be transferred to rW is stored in the cell indicated by the pointer.

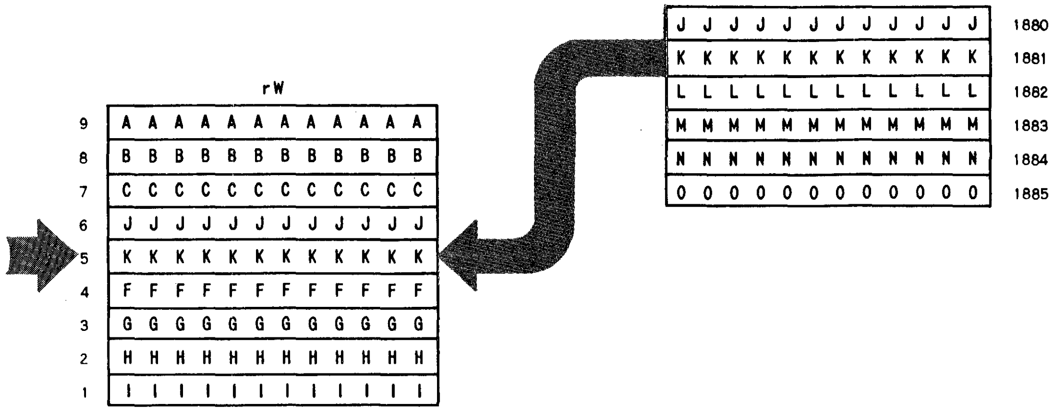

rW
9 A A A A A A A A A A A A
8 B B B B B B B B B B B B
7 C C C C C C C C C C C C
6 J J J J J J J J J J J J
5 K K K K K K K K K K K K
4 F F F F F F F F F F F F
3 G G G G G G G G G G G G
2 H H H H H H H H H H H H
1 I I I I I I I I I I I I
J J J J J J J J J J J J 1880
K K K K K K K K K K K K 1881
L L L L L L L L L L L L 1882
M M M M M M M M M M M M 1883
N N N N N N N N N N N N 1884
0 0 0 0 0 0 0 0 0 0 0 0 1885

FIGURE 8-6

This process continues until the number of words specified by the second instruction digit of the Vnm or Wnm instruction have been transferred, at which time the pointer will indicate cell one.

The following demonstrates the effect of a V6m instruction followed by a W8m instruction.

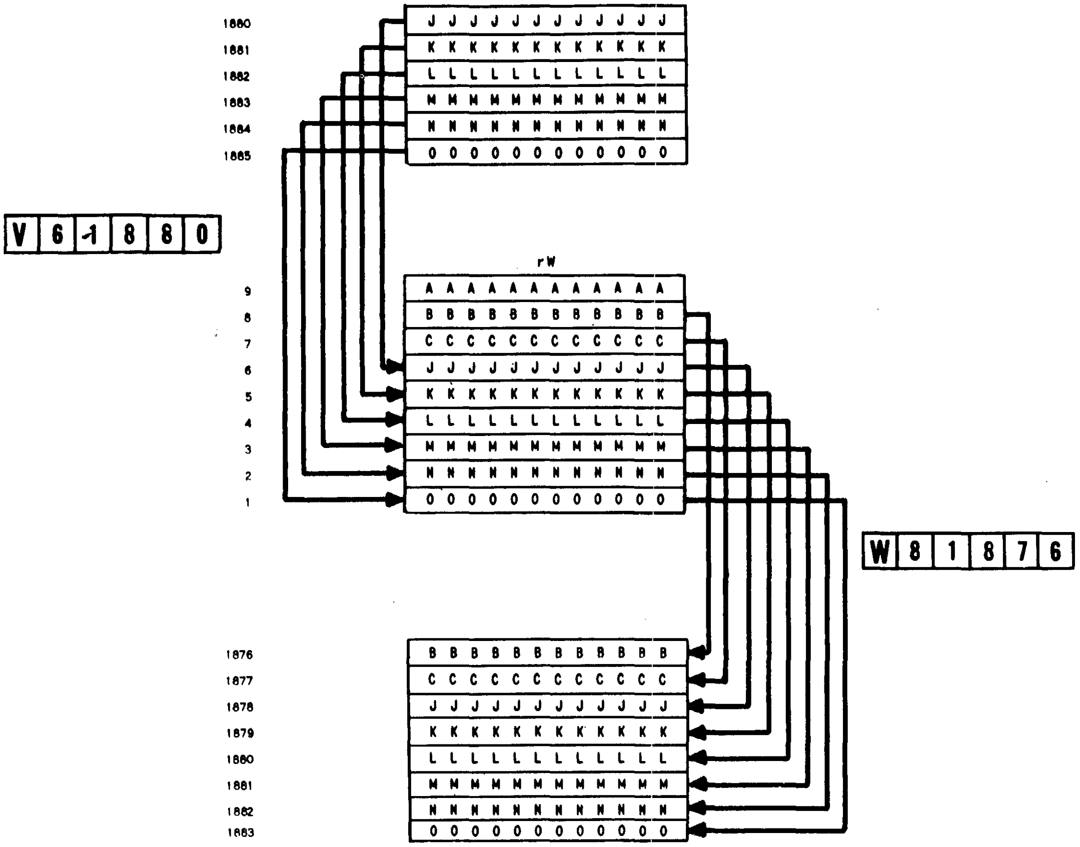

1880 J J J J J J J J J J J J
1881 K K K K K K K K K K K K
1882 L L L L L L L L L L L L
1883 M M M M M M M M M M M M
1884 N N N N N N N N N N N N
1885 0 0 0 0 0 0 0 0 0 0 0 0
V 6 -1 8 8 0
rW
9 A A A A A A A A A A A A
8 B B B B B B B B B B B B
7 C C C C C C C C C C C C
6 J J J J J J J J J J J J
5 K K K K K K K K K K K K
4 L L L L L L L L L L L L
3 M M M M M M M M M M M M
2 N N N N N N N N N N N N
1 0 0 0 0 0 0 0 0 0 0 0 0
W 8 1 8 7 6
1876 B B B B B B B B B B B B
1877 C C C C C C C C C C C C
1878 J J J J J J J J J J J J
1879 K K K K K K K K K K K K
1880 L L L L L L L L L L L L
1881 M M M M M M M M M M M M
1882 N N N N N N N N N N N N
1883 0 0 0 0 0 0 0 0 0 0 0 0

FIGURE 8-7

What happens when the second instruction digit of a Ynm instruction is different from the second instruction digit of the Znm instruction that follows it can be determined in a similar way. The cells of rZ can be thought of as being numbered 1-60 from the bottom to the top. When a Ynm or a Znm instruction is executed a pointer is set to the cell indicated by 10n. The first word to be transferred to, or from, rZ will be, or is, stored in the cell indicated by the pointer. The pointer then moves to the next lower numbered cell, and the second word to be transferred is stored in the cell indicated. This process continues until the number of words specified by the second instruction digit of the Ynm or Znm instruction have been transferred, at which time the pointer will indicate cell one.

A zero in the second instruction digit of a multiword transfer instruction and a seven, eight or nine in the second instruction digit of a Ynm or Znm instruction causes the instruction to be interpreted as a skip. Any character other than 0-9 in the second instruction digit of a multiword transfer instruction causes the computer to stall and light a neon indicating that an instruction has been improperly coded.

The attempt to execute an instruction, the address part of which is 2000 or more, causes the computer to stall and light a neon indicating that an instruction has been imporperly coded. However, the execution of a multiword instruction, which does not specify, but does imply, an address of 2000 or more, will result in a "circling of the memory". For example, if the instruction V71998 is executed, the words transferred to rW will be the contents of cells 1998, 1999, 0000, 0001, 0002, 0003 and 0004.

ARITHMETIC INSTRUCTIONS

ADD INSTRUCTIONS

Some details of the add instructions have been given on page 88 In digit positions 2-12, the characters, minus, apostrophe, ampersand and left parenthesis, are treated by add instructions, not as alphabetics, but as numerics. The minus is usually treated as a minus one (see the following illustration); the apostrophe, as a plus ten; the ampersand, a plus 11; and the left parenthesis, plus 12.

$$\begin{array}{cccccc} & - & & - & & \& \\ + & 6 & + & - & + & 5 \\ \cline{2-2}\cline{4-4}\cline{6-6} & 5 & & \Delta & & 16 \end{array}$$

SUBTRACT INSTRUCTIONS (for review, see page 43).

All rules pertaining to add instructions hold for subtract instructions. During the execution of a subtract instruction the computer changes the sign of the word being transferred from the cell specified to rX. Specifically, if the computer finds a zero in the sign position of the word, it changes it to a minus; if it finds a minus, it changes it to a zero. Actually, the computer effects this change as follows. The first two rows in figure 2-18 form pairs of characters in each column; the next two rows form other pairs of characters in each column; and so on. The characters, zero and minus constitute a pair; A and B constitute a pair; and so on. No matter what character the computer finds in the sign position it changes it to the paired character. Thus, a minus becomes a zero, and a zero becomes a minus. Likewise, an A becomes a B, and so on. If cell 1880 contains

B12345678901

and the instruction

S01880

is executed, rX will contain

A12345678901

MULTIPLY INSTRUCTIONS (for review, see pages 48, 49).

The computer performs multiplication by repeated addition. This principle can be exemplified as follows.

$$7(8) = \overbrace{8+8+8+8+8+8+8}^{\text{7 times}} = 56$$

Because each addition requires a given period of time, the computer conserves multiplication time by first building three times the value of the multiplicand and using the resulting quantity in the repeated addition.

$$3(8) = 24 \longrightarrow (rF)$$
$$7(8) = 24 + 24 + 8 = 56$$

In this manner, the computer saves the time required to perform four additions when multiplying by seven. The number of additions required by each numeric multiplier are as follows.

MULTIPLIER (rX)	NUMBER OF ADDITIONS
0	0
1	1
2	2
3	1
4	2
5	3
6	2
7	3
8	4
9	3

In the computer, the multiplicand is stored in rL. Thus, the computer builds up three times (rL) and transfers this quantity to rF for storage. Since three times (rL) may be a 12 digit number, it occupies an entire word and it has no sign. Thus, rF only contains the absolute value of three times (rL). To conserve multiplication time, the programmer should, whenever possible, treat the word requiring the fewest additions as the multiplier.

In the sign position of a word entering into a multiplication any character other than a minus is treated as a plus sign, and the product will have the proper sign in the sign position. In digit positions 2-12 the product of two characters is as shown below.

MULTIPLICATION TABLE

MULTIPLIER				MULTIPLICAND															
				i	Δ	–	0	1	2	3	4	5	6	7	8	9	’	&	(
				r	,	.	;	A	B	C	D	E	F	G	H	I	#	¢	“
				t	”	|	)	J	K	L	M	N	O	P	Q	R	$	*	?
				Σ	β	:	+	/	S	T	U	V	W	X	Y	Z	%	=	
i	r	t	Σ	25	4Δ	3	0	13	26	39	52	65	78	91	104	117	130	143	156
Δ	’	”	β	22	4(	2	0	14	28	42	56	70	84	98	112	126	140	154	168
–	.	|	:	35	50	1	0	15	30	45	60	75	90	105	120	135	150	165	180
0	;	)	+	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	A	J	/	i	Δ		0	1	2	3	4	5	6	7	8	9	10	11	12
2	B	K	S	’	(	Δ	0	2	4	6	8	10	12	14	16	18	20	22	24
3	C	L	T	7	10	i	0	3	6	9	12	15	18	21	24	27	30	33	36
4	D	M	U	4	1Δ	(	0	4	8	12	16	20	24	28	32	36	40	44	48
5	E	N	V	1	1(	11	0	5	10	15	20	25	30	35	40	45	50	55	5’
6	F	O	W	14	20	’	0	6	12	18	24	30	36	42	48	54	60	66	72
7	G	P	X	11	2Δ	9	0	7	14	21	28	35	42	49	56	63	70	77	84
8	H	Q	Y	1Δ	2(	8	0	8	16	24	32	40	48	56	64	72	80	88	96
9	I	R	Z	21	30	7	0	9	18	27	36	45	54	63	72	81	90	99	108
’	#	$	%	2Δ	3Δ	6	0	10	20	30	40	50	60	70	80	90	100	10’	11’
&	¢	*	=	2&	3(	5	0	11	22	33	44	55	66	77	88	99	110	11&	12(
(	“	?		28	40	4	0	12	24	36	48	60	72	84	96	108	120	132	144

FIGURE 8-8

THE DIVIDE INSTRUCTION (for review, see page 49).

In the sign position of a word any character other then a minus is treated as a plus sign. In digit positions 2-12 any character, regardless of whether or not it is a number, is treated as the number in its row. (See figure 2-18, i.e., M=4 in division).

ONE DIGIT AND TWO DIGIT INSTRUCTIONS

It has been stated that the function of the first and second instruction digits is to represent the operation to be performed. Some instructions represent the operation in one digit; some, in two. The former can be called one digit instructions; the latter, two digit instructions.

Two digit instructions represent the operation in the first and second instruction digits; one digit instructions, in the first instruction digit. Of the instructions covered thus far, the AFm, AHm, BFm, DFm, EFm, LFm, MFm, NFm, PFm, SFm, Vnm, Wnm, Ynm, Znm, 00m, .nm, ;nm, -nm, 0nm, and 50m instructions are two digit instructions; all others are one digit instructions. The character placed in the second instruction digit position of a one digit instruction is immaterial as long as it is not such that it changes the instruction from a one digit instruction to a two digit instruction. A K7m instruction is the same as a KOm instruction. By custom, if a particular digit is not desired in the second instruction digit of a one digit instruction, a zero is placed there. However, it is a common coding practice not to write a second instruction digit zero. For example, B01880 would be written as B 880, but still recorded as B01880.

OVERFLOW

The sum of two numbers with eleven significant integers in each will be a twelve integer number if a carry is produced. If a decimal point immediately precedes the most significant digit of each number, the carry is a whole number. In the computer this carry would go into the sign position, but this position is occupied by the sign. The computer makes the assumption that the absolute value of all quantities is less than one by preventing a carry into the sign position. An attempted carry into the sign position is called overflow.

Overflow can occur in arithmetic operations other than addition. In subtraction, only if a negative number of eleven significant integers is subtracted from a positive number of eleven significant integers can there be overflow.

$$\begin{array}{r} +\,.50000000000 \\ \underline{-\,(\,-\,.50000000000)} \\ 1\,.00000000000 \end{array}$$

Division in which, as far as the computer is concerned, the absolute value of the dividend is larger than the absolute value of the divisor causes overflow, because the quotient would be greater than one.

$$\frac{-\,.60000000000}{-\,.30000000000} = 2.0000000000$$

Similar reasoning guarantees that, in general, multiplication cannot cause overflow, since two fractional quantities must produce a fractional product. There are certain uncommon exceptions to this last statement which arise because it is possible to symbolize, in only eleven digit positions, a quantity which is greater than one by using the characters ', & and (.

Overflow occurs during γ or δ Time. The carry into the sign position is lost. If overflow occurs on γ Time, δ Time will be executed. At the end of the cycle during which overflow occurred, six zeros rather than the(CC) are transferred to SR. Therefore, on the next three stage cycle, the pair of instructions in cell 0000 are executed. On the succeeding three stage cycle, control returns to the pair of instructions in the cell specified by (CC). But (CC) were one greater than the address ofthe line being executed when overflow occurred, and since then an additional three stage cycle has been completed,again increasing (CC) by one. If overflow occurred due to an instruction in cell k, then the instructions in cell k + 2, now specified by the present (CC), will be executed; provided that neither cell k, nor cell 0000, contains a transfer of control instruction.

Consider how this principle might be employed in programming. Addition is sometimes used for purposes other than summation. One of these uses is to alter addresses in an iterative routine. For example, with a series of two word items, where the first word is a social security number, the next social security number may be selected by adding two to the address of the current social security number. There will be a limit to the number of these social security numbers with which it is necessary to deal. When the limit is reached the computer must take some other action.

By adding to a word each time the address is advanced, overflow will eventually occur. The number of addresses that have been advanced can be counted by this addition. Suppose that after processing sixty words it is necessary to take some other action. With a two word item there will be 30 items. If a 70 is placed in the 2nd and 3rd digit positions of the word used as a counter, and 1 is added in the 3rd digit position each time 2 is added to the address, overflow will occur after the 30th item has been processed, since 70 + 30 produces a carry. If the add order occurred in cell k, then the instructions for taking the new course of action would be stored in cell k+2. When overflow occurs control will be transferred to this cell.

The add order, as has been pointed out, is being used to advance the address part of an instruction. It will not be necessary to have another add order to increase the word used as a counter. There are two instructions per word, and only one instruction has been used for the transfer order. The other instruction in this same instruction line may be used as the counter. At the same time the address part is being advanced, the item counter can be advanced by adding to the appropriate digits of the same instruction line. However, the first digit position of the counter must be either a part of the program, or an instruction which will not adversely effect the program. A superfluous KOm instruction may be used, for example, if (rA) and (rL) are of no concern. The variable word which contains the counter and the variable address might initially have the following appearance.

K70000V21882

and the following constant could be added to it.

001000000002

In summary, overflow permits an alternate course of action based on the decision, "have all the items in the set been processed?" The routine to which transfer is to be made on reaching this limit will begin with the instructions in cell k + 2. The instructions in cell k + 1 will relate to the normal item advance routine. The instructions in cell k + 2 will be reached only on overflow, and instructions in cell k + 1 will not be executed when overflow does occur.

Consider the following example.

Each of a set of 30 two word items is to be processed. The items will be processed in a working storage. The problem is to replace the contents of the working storage with successive items of the set and when the set (cells 1880 - 1939), is exhausted, transfer control to some other cell where an ending routine is coded.

Without utilizing overflow the item advance routine might be as follows.

```
0020   B00022
                    L00024  )
0021   A00025               }  i:30
                    Q00015  )  to ending routine
---------------------------------
0022  [V21882              ]
      [             W21880 ]
0023   C00022                  i + 1 ──► i
                    U00002     to processing
---------------------------------
0024   V21942               )
                    W21880  }  Constants
0025   000002       ↪       )
```

In this coding there is one section identified with the decision i:30 and a separate section for the operation i + 1 ──► i.

Employing overflow in the coding below there remains a subroutine associated with the operation i + 1 ──► i. However, the coding for the decision i:30 is not obvious. The decision i:30 is incorporated into the coding of the operation i + 1 ──► i by taking advantage of the effect of overflow.

```
0000     ↪
                   ↪
 .             .
 .             .
 .             .
0020     K70000
                   V21882
0021     B00026                  )   i + 1 ──► i   (cell k)
                   AH0020 *      )
0022     W21880                                    (cell k + 1)
                   U00002            to processing
--------------------------------
0023                             )                 (cell k + 2)

0024                             }   ending subroutine

0025                             )

0026     001000                      constant
                   000002
```

The asterisk in the remarks column indicates that overflow is being used as a control.

The routine operates as follows. When control initially reaches the item advance routine, the superfluous KOm instruction is executed on γ Time. On δ Time the contents of cells 1882 and 1883 are transferred to rW. The address in CC is now 0021, and the word in this cell will be the next executed. On γ Time the constant for advancing the counter and the address - the contents of cell 0026 - is transferred to rA. On δ Time (rA) and (0020) are added, and the sum is transferred to cell 0020, so that it now contains

K71000V21884

(CC) now contains 000000000022 and the instruction pair in cell k + 1 will be the next executed. On γ Time the second item is transferred to working storage. On δ Time control is transferred to processing. Afterprocessing the second item, control once more returns to the item advance subroutine. Each iteration through the item advance subroutine operates as described above with the result that the contents of cell 0020 are successively

K72000V21886
K73000V21888
K74000V21890

and so on, until the 30th item is processed. At that point the contents of cell 0020 are

K99000V21940

After processing the 30th item control returns to the item advance subroutine. The K0m instruction is executed. The contents of cells 1940 and 1941 are transferred to rW.* The contents of cell 0020 are transferred to rA. The execution of the AHm instruction adds one to the 99 in the second and third digit positions of the contents of cell 0020, and overflow occurs. The carry is lost, six zeros are transferred to SR, the skips in cell 0000 are executed, and control returns to the cell specified by (CC). When overflow occurred on line 0021 (or k), (CC) were 0022. Prior to the execution of (0000), this reading was increased by one, so that (CC) now reads 0023 (or k + 2). Consequently, control goes to the ending subroutine.

In the above example, the end of the set of items was determined by counting the items as they were processed. The end of the set can also be determined by using overflow to "test" the address of the item currently being processed against the address of the last item.

*Note that if the data had been stored in cells 1940-1999 the last V2m order, V22000, would cause the computer to stall because the address is non-existent.

```
0000

 .           .
 .           .
 .           .
0020    V21882
                    B00026     }   i + 1 → i
0021    AH0020
                    A00027 *       i : 30
0022    W21880
                    U00002         to processing
__________________________
0023
0024                                ending subroutine
0025

0026    000002
0027    078058                      constants
```

This coding operates in essentially the same way as the previous example. When the 30th item is being processed cell 0020 will contain

V21940B00026

After the 30th item has been processed, control returns to the item advance subroutine. The contents of cells 1940 and 1941 are transferred to rW. The constant

000002000000

is transferred to rA. The contents of cell 0020 are added to (rA), and the sum

V21942B00026

is transferred to rA and cell 0020. The constant

078058000000

is added to (rA), causing overflow and thus directing control to the ending subroutine.

The above are examples of "generalized overflow" with an increment of one. The overflow is called generalized, because no matter in what cell, k, overflow occurs, control is always directed to the contents of cell k + 2. The overflow is said to have an increment of one, because overflow always directs control to the contents of the cell whose address is one plus the address of the cell to which control would otherwise pass.

Overflow can be used in a special, as well as in a general, sense. If a 90m instruction were stored in cell 0000, the use of overflow would be special rather than general, since in no matter what cell overflow occurs, the result is some specific operation, namely, the computer stops.

By means of a coded generalized overflow routine, generalized overflow with an increment other than one can be used.

```
0000    R00024
                  U00023
 .                .
 .                .
 .                .
0020    K70000
                  V21882
0021    B00025                  }
                  AH0020 *      }  i + 1 → i
0022    W21880
                  U00002           to processing
0023    B00026                  }
                  AH0024        }  generalized overflow with increment
0024    └──┐                    }  of five
                  UOVAR         }
0025    001000                  }
                  000002        }  constants
0026                            }
                  000004
0027                            }
                                }
0028                            }  ending subroutine
                                }
0029                            }
```

This coding is identical to the first example in this chapter except that it uses generalized overflow with an increment of five. When overflow occurs, (CC) are

000000000022

On β Time (CC) are increased by one. The execution of the ROm instruction transfers the word

000000U00023

to cell 0024. The UOm instruction transfers control to cell 0023. The constant

000000000004

is transferred to rA. The contents of cell 0024 are added to (rA), and the sum

000000U00027

is transferred to cell 0024. On the next three stage cycle the 00m U0m instruction pair just fabricated is executed, transferring control to cell 0027, the address of which is five more than the address of the cell to which control would have passed if overflow had not occurred.

This coding can be used for generalized overflow with any increment desired. The only variant is the constant used in the generalized overflow subroutine, this constant always being one less than the value of the increment desired.

The only caution that must be observed in the use of generalized overflow is as follows. No matter what increment, n, is used, and no matter where and how many times overflow is used for control purposes, the subroutine to be followed when overflow occurs must be coded n + 1 cells below the cell in which overflow occurs.

UNDESIRED OVERFLOW

There are many uses of arithmetic instructions in which the unplanned occurrence of overflow would result in an incorrect solution. Although the occurrence of overflow can not be prevented, a minus sign coded in the second instruction digit of an instruction on which overflow occurs will stop the computer on the completion of the execution of the instruction.

STUDENT EXERCISES

Utilize overflow as a control.

1. Reading the data stores 60 one word quantity items of form

$$000000QQQQ_{\wedge}QQ$$

in cells 1880-1939.

a. Print the sum of the quantities.
b. Print the sum of the quantities and the subtotal of the first ten quantities, the subtotal of the next ten, and so on, up to and including the subtotal of the last ten.

2. Reading the data stores 60 one word credit account number items of form

   0AAAAAAAAAAA

   in cells 1820-1879, and 60 one word delinquent account number items of form

   0DDDDDDDDDDD

   in cells 1880-1939. Write 60 one word credit items of form

   KAAAAAAAAAAA

   where A - credit account number

   K - credit key, and may take values
   G - credit good
   B - no credit.

3. Reading the data stores six ten word A items in cells 1820-1879 and six ten word B items in cells 1880-1939. The first word of each item is a key. The A and B items are each arranged in ascending order by key. Write the merged items.

SUMMARY

TRANSFER OF CONTROL INSTRUCTIONS

The UOm, QOm, and TOm may be coded as left hand instructions providing that the address, m, is the address of the right hand instruction. Thus a Q-T pair of instructions is legitimate and will transfer control if (rA) $\geq$ (rL).

MULTIWORD TRANSFER INSTRUCTIONS

If the pair of instructions $Vn_1 m_1$, Wn_2m_2 ($Yn_1 m_1$, Zn_2m_2) is executed and $n_1 \neq n_2$ the following results:

IF

$n_1 > n_2$ The first $n_1 - n_2$ (tens of) words transferred to rW (rZ) are not transferred to m_2.

$n_1 < n_2$ The $n_2 - n_1$ (tens of) words transferred to rW (rZ) by a previous instruction are transferred to m_2, followed by the (ten) n_1 words of the current instruction.

SHIFT INSTRUCTIONS

Onm: Error if n ≠ 0 to 9, skip if n = 0.
. nm, –nm, or ;nm: error if n ≠ 1 to 9.
(rA) are unaltered by the error.

ARITHMETIC INSTRUCTIONS

The following tables illustrate the effect of adding any combination of characters.

N: 1 thru 9, ', &, or (. ',&, and)are treated as 10, 11, and 12,respectively.
C: any character except 0, –, or "N" (as defined above).
S: the sign of the larger in absolute value of the two words being added.
Δ: Space.

A. For any position except the sign

	N	C	0	–
N	Normal Sum	C	N	N-1
C	⁻ C	Error	C	C
0	N	C	0	–
–	N-1	C	–	Δ

B. In the sign position

	N	C	0	-
N	Error	Error	N	N
C	Error	Error	C	C
0	N	C	0	S
–	N	C	S	-

In the S0m and S1m orders the character in the sign position is changed to its paired character (cf. p.153). The rules above then apply.

Multiplication is performed by repeated additions.3 (rL) is stored in rF and used in this process. Overflow will occur while forming3 (rL) if one of the characters ' & or (is the most significant digit. Multiplication by non-numeric characters is shown in figure 8-8.

Division is performed by repeated additions and subtractions, but will always produce a numeric quotient even if the dividend and/or divisor contain non-numeric characters. Overflow will occur if (rL) $\leq$ (m).

Overflow occurs when there is an attempted carry into the sign position during an arithmetic order. Following the cycle during which overflow occurred, (CC) are stepped, and (0000) are executed. The instructions which can cause overflow are the A0m, AHm, D0m, M0m, N0m, P0m, S0m, X0m, as well as these when used as field selection instructions.

Overflow is used as a control as follows:

A. To stop the computer: if a minus sign is placed in the second instruction digit position of the instruction causing overflow.

B. Specialized: to perform one particular operation, or series of operations, regardless of where overflow occurs.

C. Generalized: to perform any number of distinct operations depending on where overflow occurs.

If the instruction causing overflow i's in cell k and:

1. if (0000) is a pair of skip instructions, the instructions in cell k + 2 will be executed following the execution of the skips.

2. if (0000) is R0(L + 1) U0L and if the following instructions are stored in cells L, L + 1, and L + 2

```
L        B0 (L + 2)
                        AH (L + 1)
L + 1   [EXIT                     ]
                        LINE
L + 2
                        00000n
```

the sequence of instructions following those in cell k will be0000, L, L + 1, L + 2 and k + 2 + n.

chapter 9

Input – Output

Magnetic tape is the means of introducing, and removing, large volumes of data to, and from, the memory, The tape may be metal or plastic, both being about one half inch wide and .002 inches thick. Data may be written on a tape, read and erased, and new data written on the same tape reliably over 1000 times, thus cutting the cost of supplies. Magnetic tape comes in various lengths, the longest metallic tape being about 1550 feet; plastic, about 2400 feet.

Characters are recorded on tape in coded form. The code for each character consists of a unique combination of magnetic and non-magnetic spots. The characters are recorded on the tape serially, and the coded bits of any one character are recorded in parallel.

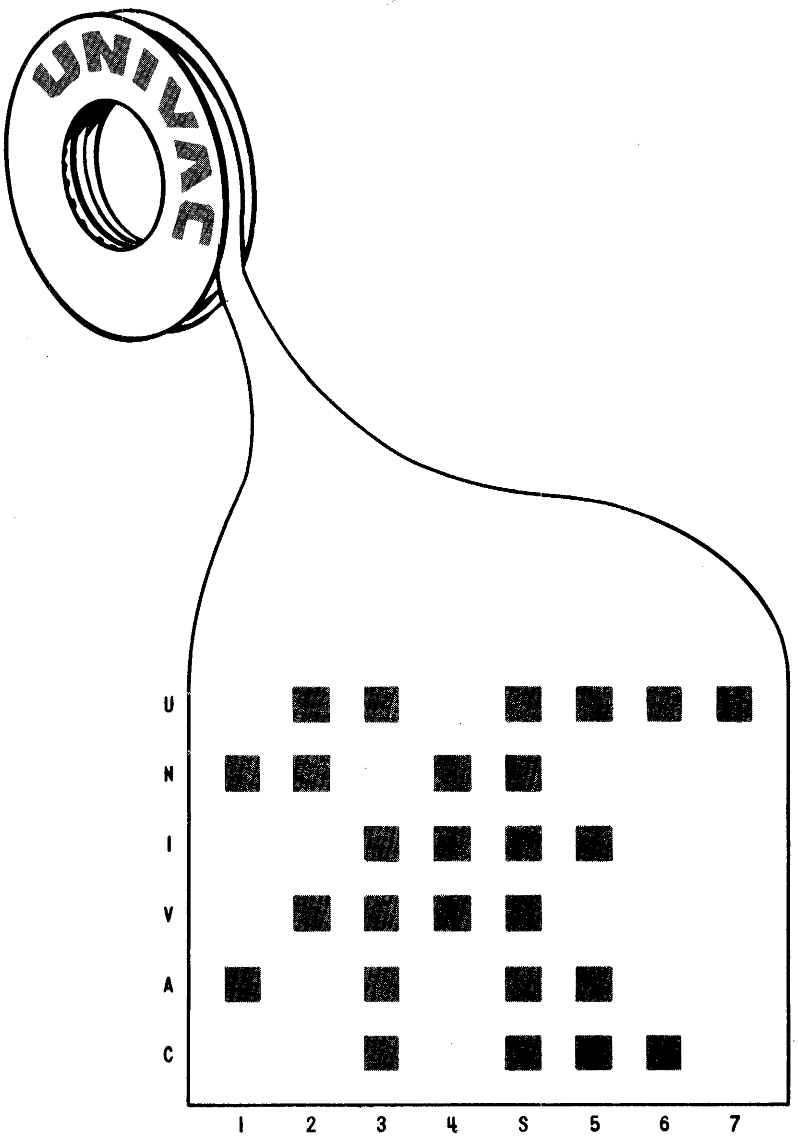
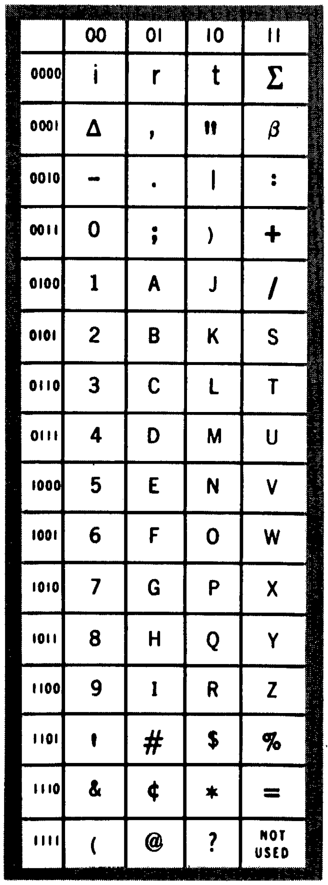


FIGURE 9-1

CHARACTER REPRESENTATION

The code for each character can be represented as a series of ones and zeros,referred to as bits,and corresponding to the magnetic and nonmagnetic spots on tape. The basic representation of each character is given in the following figure.



	00	01	10	11
0000	i	r	t	Σ
0001	Δ	,	"	β
0010	-	.	|	:
0011	0	;	)	+
0100	1	A	J	/
0101	2	B	K	S
0110	3	C	L	T
0111	4	D	M	U
1000	5	E	N	V
1001	6	F	O	W
1010	7	G	P	X
1011	8	H	Q	Y
1100	9	I	R	Z
1101	'	#	$	%
1110	&	¢	*	=
1111	(	@	?	NOT USED

CODE COMBINATIONS OF
THE 63 UNIVAC II CHARACTERS

FIGURE 9-2

In the basic representation, from left to right, the zone of the character precedes the excess three portion. Thus,

010100

is the basic representation of the character A.

Electronically, there is the possibility of gaining or losing a one in a bit position when a character is transferred from one storage to another. To check for such an occurrence, an extra bit position, called the check bit position, precedes the basic representation of each character. The basic representation of a character may contain an odd or even number of ones. Those characters whose basic representation contains an even number carry a one in the check bit position; those with an odd number, a zero. When a character is transferred, the ones in its representation are counted. If an even count results, a one has been gained or lost, and an error, called the odd-even error, has occurred. The occurrence of an odd-even error stalls the computer and lights an appropriate neon.

Thus,

1010100

is the representation of the character A,

0000100

the character one.

When a character is written on tape, one additional magnetic spot, called a sprocket pulse, is recorded for checking purposes.

THE UNISERVO

The Uniservo is the device by which the computer reads from and writes on tape. The Uniservos are named 1 thru 9, –, and A thru F.

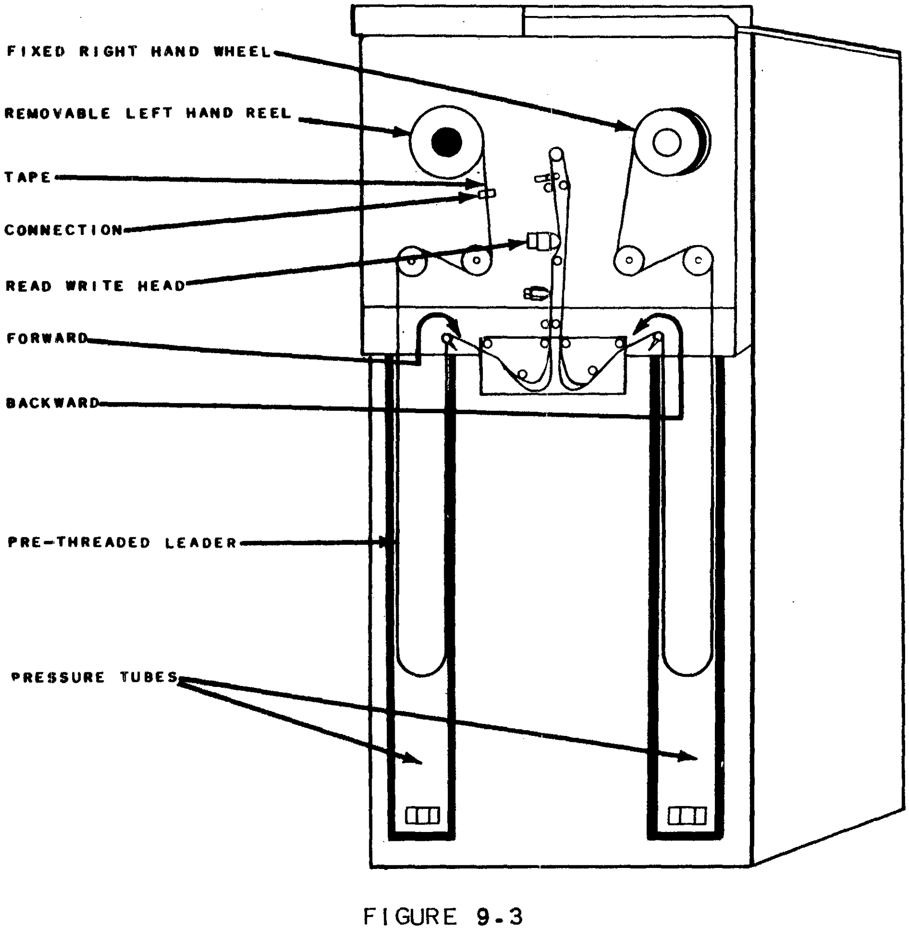


FIGURE 9-3

Since the right hand reel is permanently fixed, a tape to be read from or written on is mounted on the left hand reel. The tape is connected to a pre-threaded leader which is fastened to the right hand reel. Because of the pre-threaded leader, removal of a reel and the mounting of a new reel takes only one half minute.

Since characters are written on tape serially, the meaning of the characters depends on the sequence in which they were written, just as the meaning of the frames on a movie reel depends on the sequence in which they were shot. The permanently fixed right hand reel guarantees that, when a tape is mounted, the characters on the tape are in the sequence in which they were written.

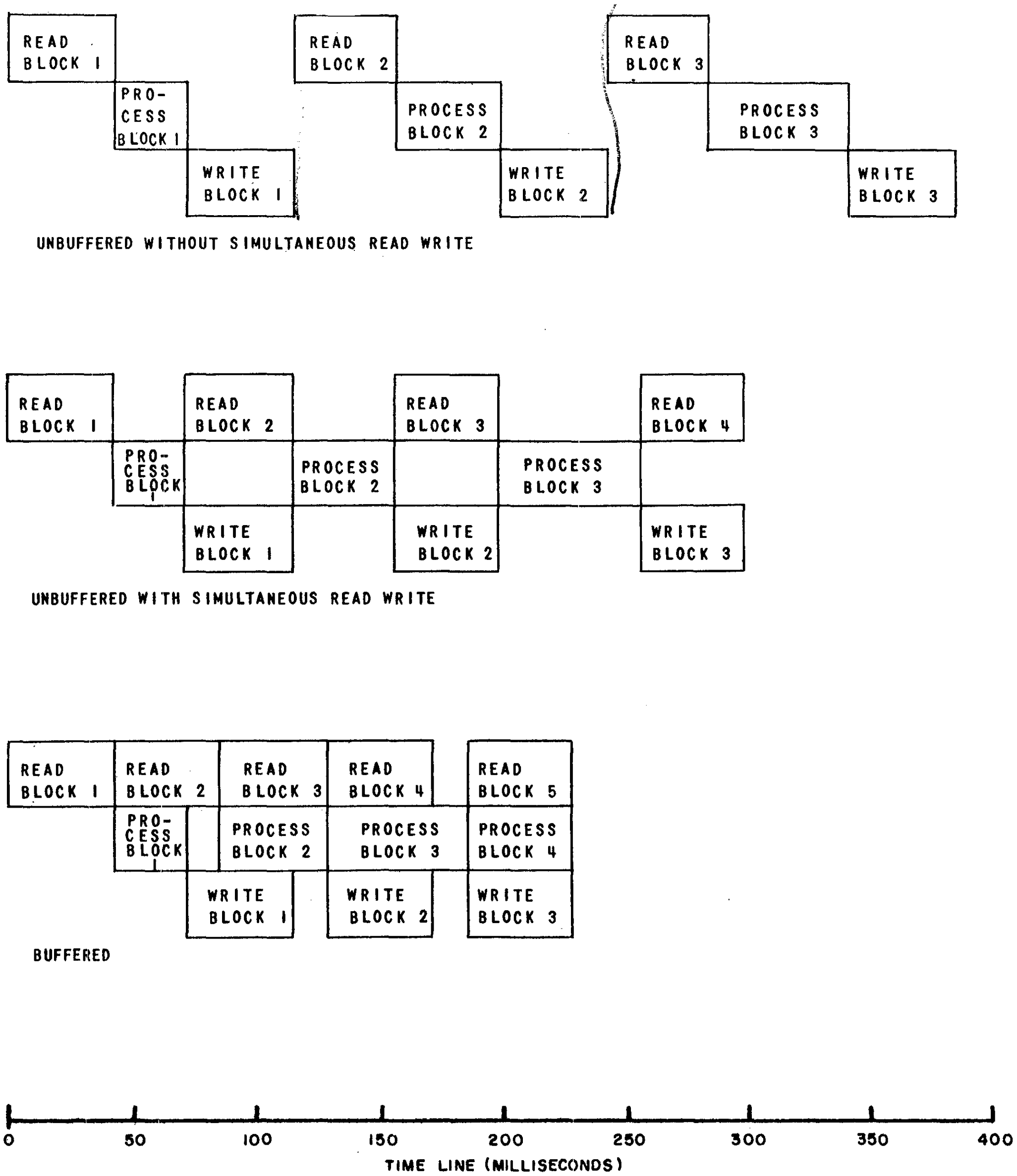
When tape is passing from the left hand to the right hand reel, the tape is said to be moving forward; from right to left, backward.

THE BLOCK

To reduce the amount of time required for starting and stopping tapes, data is grouped into units called blocks. A block is the unit of data that the computer reads or writes with the execution of a single instruction and is composed of 60 words.

BUFFERING AND BACKWARD READ

Data is processed by the Univac Central Computer at electronic speed. Computer processing time may be increased by the relatively slow electro-mechanical means employed to provide input and output. Transfer of data from tape to electronic storage is not as rapid as transfer from one electronic storage to another, but to overcome this, simultaneous read-write features are employed. A comparison of a system incorporating the simultaneous read-write feature with a system not incorporating this feature is shown in figure 9-4. By providing a system of reservoirs, called "buffers", which hold a reserve of data, a delay in processing is avoided by parallel operation. The Uniservos work simultaneously with the computer, thus enabling tapes to be written, read, and rewound at the same time that the computer is processing. A comparison of a completely buffered system with a system incorporating the simultaneous read-write feature in shown in figure 9-4.



UNBUFFERED WITHOUT SIMULTANEOUS READ WRITE

UNBUFFERED WITH SIMULTANEOUS READ WRITE

BUFFERED

FIGURE 9-4

Many applications require more than one pass over the data. Rewind time is measured in minutes,and considerable time can be lost waiting for a tape to be rewound in order that it can be reread. If a computer can read data from a tape while the tape is moving backward, a second pass can be made without the delay for rewind. The Central Computer of the Univac System incorporates both buffers and the backward read feature.

THE BUFFERS

Data to be written is transferred from its location in the memory to register O(rO), a 60 word register. The data in rO is then transferred to a Uniservo one character at a time to be written on tape. Once rO has been filled, the computer is released to perform other operations because the separate output control circuits direct the write operation independently of the computer.

Data to be read is initially transferred character by character from tape and accumulated in register I (rI), a 60 word register. The data in rI can then be transferred to the memory. Once the transfer of data from tape to rI has begun, the computer is released to perform other operations.

The use of these registers between the computer and the Uniservos evables the computer to be held up for only the small amount of time necessary to fill the output buffer, rO, or to empty the input buffer, rI, or to initiate a read operation.

TAPE INSTRUCTIONS

"T" represents "tape", and "n" represents the Uniservo affected (1,...,9, –, A, ... , F).

INSTRUCTION	OPERATION
1nm	Tn → rI

Read a block forward from Tn to rI.

When executing the 1nm instruction, the computer ignores m.

INSTRUCTION	OPERATION
2nm	rI ← Tn

Read a block backward from Tn to rI.

When executing the 2nm instruction, the computer ignores m.

As defined, the 1nm and 2nm instructions allow the possibility of reading a given block into rI and then, without transferring the given block from rI into the memory, reading another block into rI, thus destroying the given block. To guard against such a loss of data, the computer - after executing a 1nm or 2nm instruction and before executing another 1nm or 2nm instruction-checks to see if the given block has been transferred from rI into the memory at least once. If such is not the case, the computer stalls and lights a neon to indicate the situation.

INSTRUCTION	OPERATION
30m	(rI) → m,..., m + 59

Transfer (rI) to 60 consecutive cells starting with m.

The 30m instruction is a two digit instruction.

INSTRUCTION	OPERATION
40m	(rI) → m,..., m + 59

Transfer (rI) to 60 consecutive cells starting with m.

The 40m instruction is a two digit instruction and is identical in effect to the 30m instruction.

INSTRUCTION	OPERATION
3nm	(rI) → m,..., m + 59; Tn → rI

Transfer (rI) to 60 consecutive cells starting with m.
Read a block forward from Tn to rI.

INSTRUCTION	OPERATION
4nm	(rI) → m,..., m + 59; rI ← Tn

Transfer (rI) to 60 consecutive cells starting with m.
Read a block backward from Tn to rI.

Since the forward read instructions, 1nm and 3nm, read the first word of the block first; the second word second; the third, third; and so on; until the 60th word is read last; while the backward read instructions, 2nm and 4nm, read the 60th word of the block first; the 59th word, second; the 58th, third; and so on; until the first word is read last; the question arises, how is the block stored in the 60 cells that constitute rI? The cells can be thought of as being numbered 1-60 from top to bottom. When a forward read instruction is executed, rI is filled from the top down, with the consequence that the first word of the block is stored in cell 1; the second word of the block, in cell 2; the third word, in cell 3; etc.; until the 60th word is stored in cell 60. When a backward read instruction is executed, rI is filled from the bottom up, with the consequence that the 60th word of the block is stored in cell 60; the 59th word of the block, in cell 59; the 58th word, in cell 58; and so on; until the first word is stored in cell 1. Therefore, both forward and backward read instructions store the block in rI in the same final configuration.

INSTRUCTION	OPERATION
5nm	(m, ..., m + 59) → Tn

Write the contents of 60 consecutive cells, starting with m, on Tn at 250 characters per inch.

The 5nm instruction is executed by filling rO, releasing the computer, and then writing from rO onto the tape on Uniservo n.

INSTRUCTION	OPERATION
6nm	RWD Tn

Rewind Tn.

When executing the 6nm instruction, the computer ignores m.

INSTRUCTION	OPERATION
7nm	(m, ... , m + 59) → Tn

Write the contents of 60 consecutive cells, starting with m, on Tn at 50 characters per inch.

INSTRUCTION	OPERATION
8nm	RWD * Tn

Rewind Tn; set interlock. Any subsequent instruction involving Tn stalls the computer.

When executing the 8nm instruction, the computer ignores m.

After the execution of a 8nm instruction Tn is referred to as interlocked. The function of interlock is that, once an output tape has been written and rewound, the tape is automatically protected against the possibility of another write, which would destroy the output data. Interlock is released by removing the tape from the Uniservo.

Another method used to protect information is to insert a metal snap ring in the reel of an input tape. This causes the Uniservo on which the tape is mounted to be interlocked for writing, but not for reading or rewinding, thus protecting against the possibility of a write, which would destroy the input data.

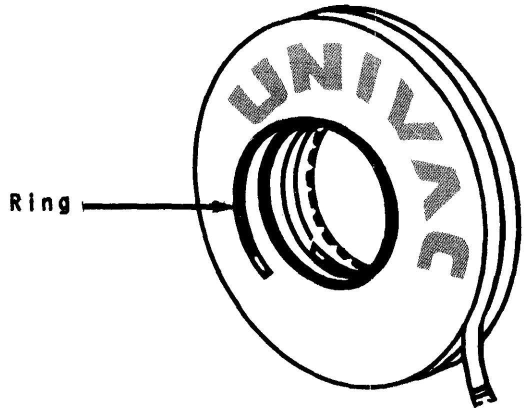


FIGURE 9.5

Essentially, the input-output orders are executed in the following steps:

1. Interlock Tests
   This step is used to determine if:
   a. the desired servo is already in use. (an input-output error has the same effect as if the servo were in use)
   b. there is another input (output) order in effect if the present order is one of input (output).

   If one of the above is true the computer waits, or is interlocked, until the interlock causing order is completed. In the case of an error the wait is relatively long, because the order cannot be completed, and will draw the attention of the computer operator.

2. Initiation of the order
   This varies for the orders so that for:
   a. 1n, 2n, 6n, or 8n, tape movement begins.
   b. 30m or 40m, (rI)are transferred to the memory, <u>completing</u> the order.
   c. 3nm or 4nm, (rI)are transferred to memory and tape movement begins.
   d. 5nm or 7nm, the block is transferred to rO.

3. Completion of the order
   The entire block is read or written, or the tape is rewound.

Steps 1 and 2 require the use of the Control Unit, while step three, the greater part of the order, takes place under the control of the input-output circuits. These steps result in the computer being able to read, write, rewind, and process at the same time.

TAPE INSTRUCTIONS ON FLOW CHARTS

There is a symbol for each tape instruction.

INSTRUCTION	EXAMPLE SYMBOL
1nm	Tj → rI
2nm	rI ← Tj
30m, 40m	rI → J
3nm	rI → J Tj → rI
4nm	rI → J rI ← Tj
5nm, 7nm	P → Tp
6nm	RWD Tj
8nm	RWD * Tj

In the flow chart Tp may be a reel of tape in file P; Tj, a reel in file J; etc.

SENTINELS

Generally the amount of data on a tape is unknown and varies from one application to the next. To determine when all the data has been processed, a sentinel convention is used. Six Z's in digit positions one through six are placed in the key of the item immediately following the last data item and in the last word of the block containing this item. Immediately following this block is a second block with the six Z's in the first six digits of the key of the first item and of the last word of the block.

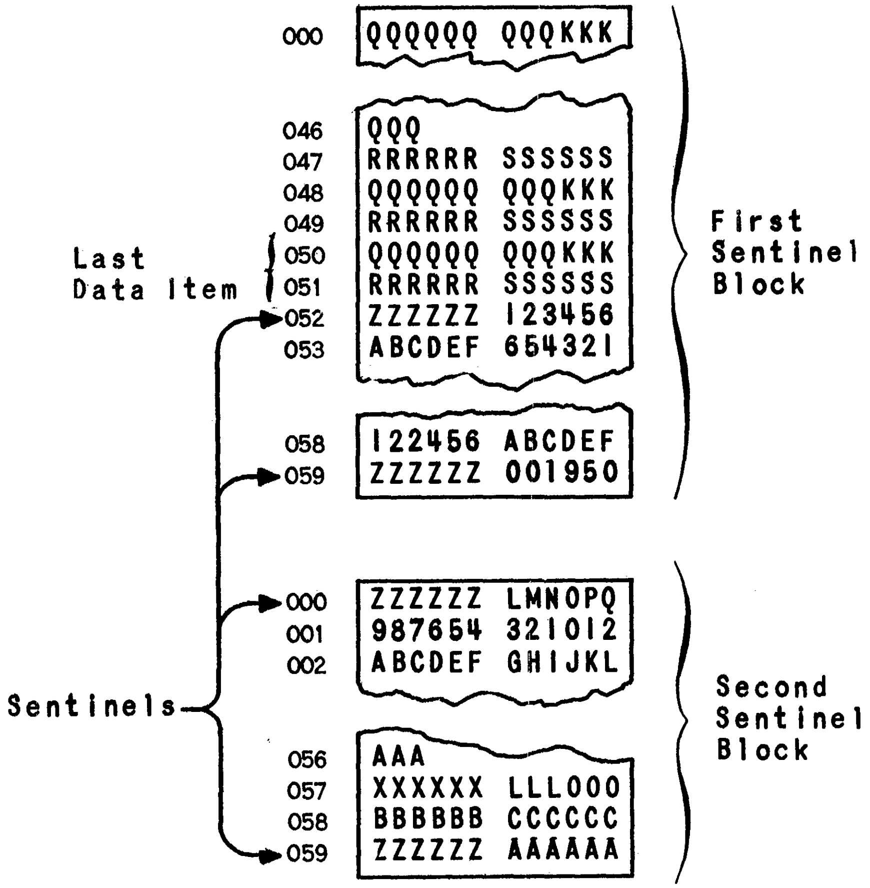


FIGURE 9-6

THE INSTRUCTION TAPE

An instruction tape may be designed to be mounted on any Univac Uniservo. For purposes of this manual Uniservo 1 will be used.

The Uniservo to be initial read is selected by a manual operation on the Supervisory Control Panel. The initial read operation reads a block from the Uniservo selected, the tape moving forward, and transfers the block to cells 0000-0059. All subsequent movements of the instruction tape are ordered by instructions stored in the memory.

SERVO DELTA

On the Supervisory Control Panel is a set of 16 buttons called Δ Tape Selector buttons and labelled with the names of the Uniservos. If a delta is coded in the second instruction digit of a tape instruction, the computer executes the instruction with respect to the Uniservo whose Δ Tape Selector button is depressed.

ILLUSTRATIVE EXAMPLE

A tape contains a series of ~~three~~ 4 word job items of form

```
NNNNNNN00000
00CCCCCLLLLL
     ^    ^
00MMMMMOOOOO
     ^    ^
```

where N - job number
C - contract price
L - labor cost
M - material cost
O - overhead cost

There is at least one full block of data on the tape.

For each job item, produce a one word profit item of form

```
NNNNNNNPPPPP
          ^
```

where N - job number
P - profit

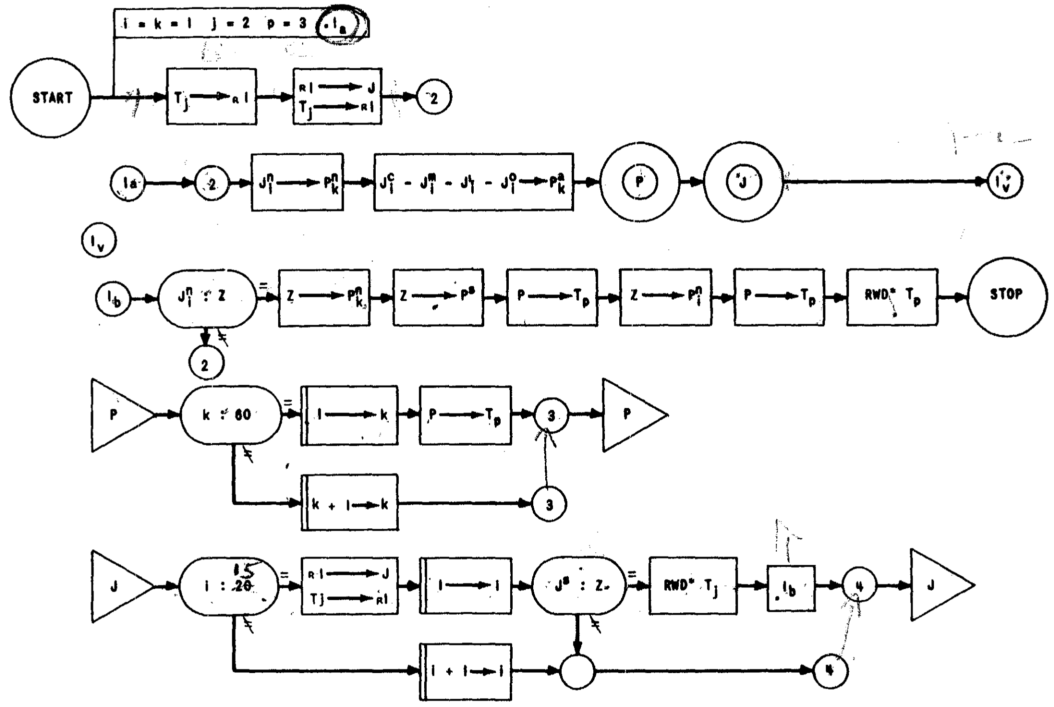
Write the profit items.

SERVO ALLOCATION

To solve the problem, Uniservos must be allocated to the input and output tapes. The servo allocation might be.

UNISERVO	TAPE
2	Job = Tj
3	Profit = T_p

FLOW CHART



LEGEND

J - SET OF JOB ITEMS

J_i - iTH ITEM IN J, i = 1,...., 20

J_i^n - NUMBER OF J_i

J_i^c - PRICE OF J_i

J_i^m - MATERIAL COST OF J_i

J_i^L - LABOR COST OF J_i

J_i^o - OVERHEAD COST OF J_i

J^s - SENTINEL OF J

P - SET OF PROFIT ITEMS

P_k - KTH ITEM IN P, k = 1,...., 60

P_k^n - NUMBER OF P_k

P_k^a - AMOUNT OF P_k

P^s - SENTINEL OF P

FIGURE 9.7

The following is a description of the thinking that might have accompanied this flow chart. (The coding for this example is on page184)

The first thing to be done is to read a block of job items from T_j into the memory. To effect this transfer, the block must first be read into rI.

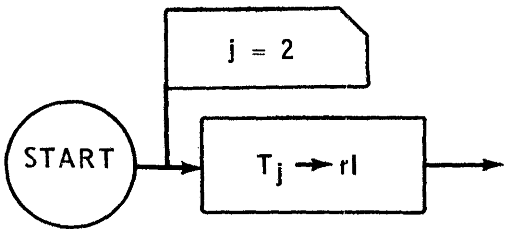


FIGURE 9.8

(rI) must be transferred to the memory. This transfer could be done with a 30m instruction. However, to take full advantage of the buffer system, while the job items stored in the memory are being processed, the next block of items should be read from tape into rI. By using the 3nm instruction this situation can be effected.

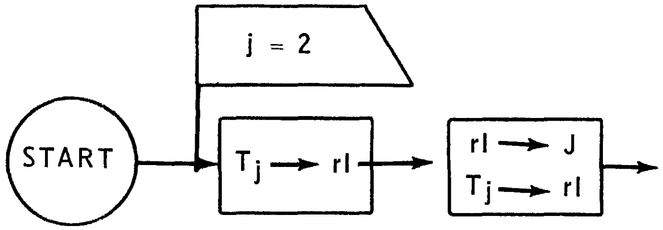


FIGURE 9.9

With a block of job items in the memory processing can begin.

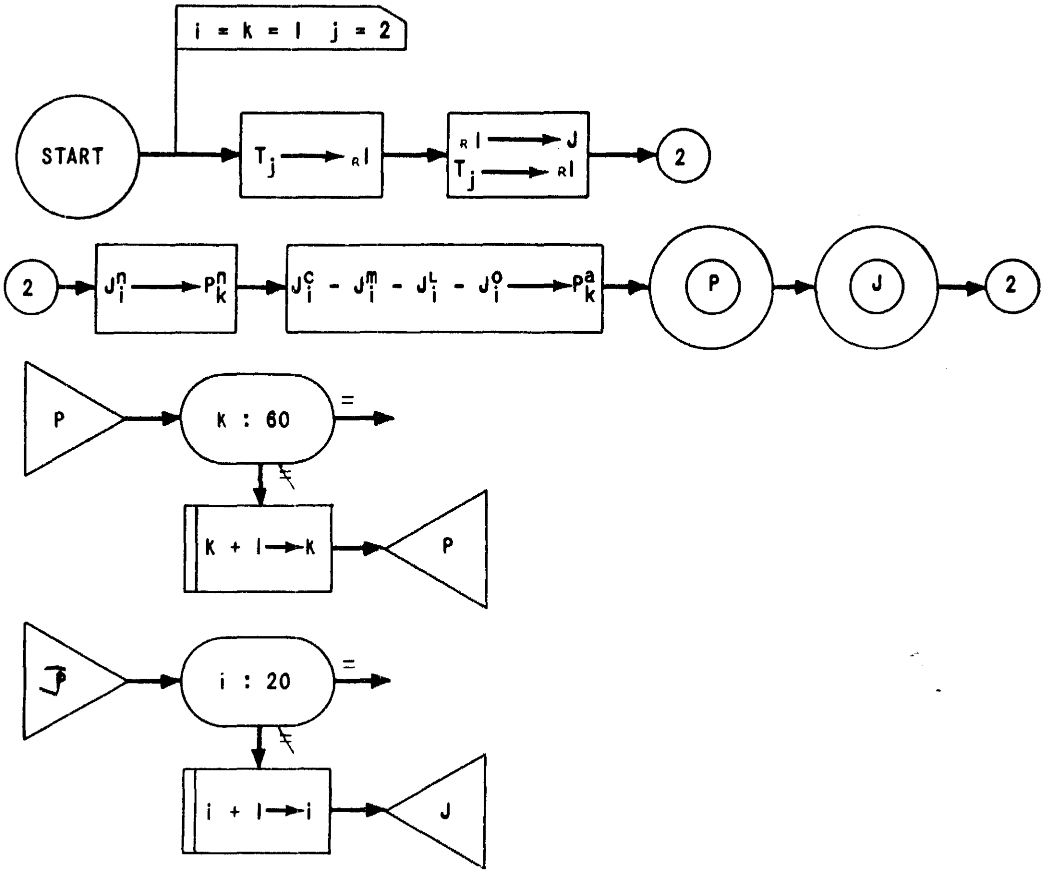


FIGURE 9.10

When a block of job items is exhausted the input item counter equals 20. To continue processing, the next block of job items, currently stored in rI, must be transferred to the memory, and the input item counter must be reset to one.

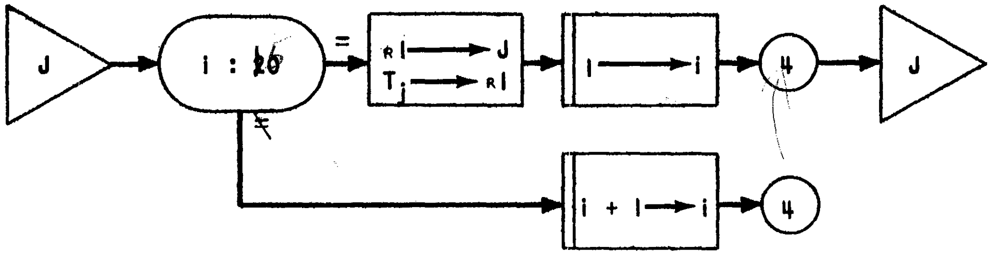


FIGURE 9.11

When the output block is filled, the output item counter will equal 60. The output item counter is reset to one to prepare for the next output block, and the current output block is written.

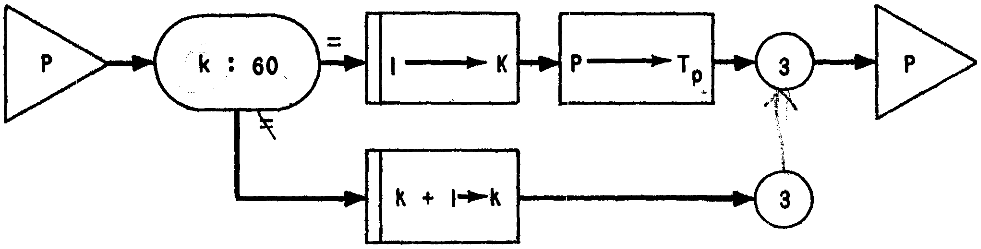


FIGURE 9-12

The only problem remaining is to determine when all of the job items have been processed. Any block of items but the first may be the last block. If it is, there will be six Z's in digit positions 1-6 of the last word of the block. If it is not, the Z's will not be present. An equality test can distinguish between the two conditions.

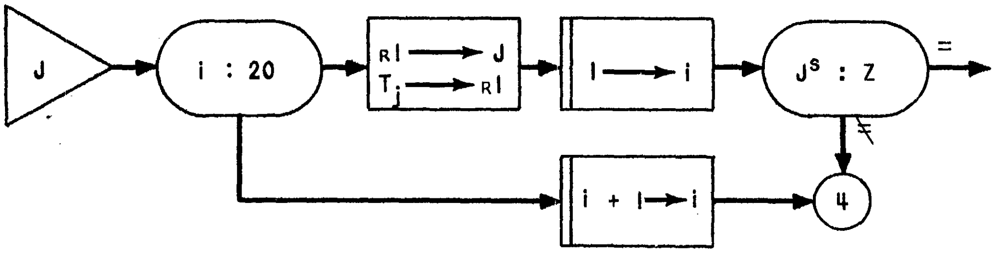


FIGURE 9.13

When the sentinel is found in the last word of the block to be processed, T_j can be rewound, and the key of each item must be tested before processing to determine whether or not it is a sentinel. A variable connector inserts this sentinel test.

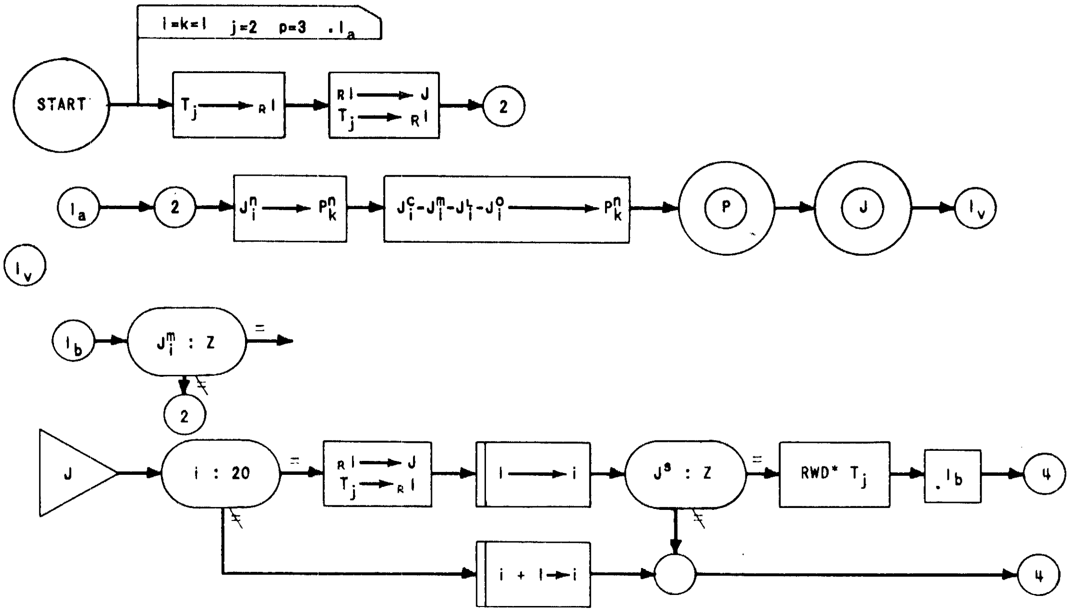


FIGURE 9-14

When the key of the item to be processed is a sentinel all the data has been processed. Sentinels must be written on Tp. The last block ofoutput is in the memory. A sentinel must be stored in the key of the item immediately following the last data item. This sentinel item must be P_k, since the output item counter always reads one more than the last item stored. A sentinel is store in P_k^N and in P^S, the last word of the block. The block is written on T_p, thus writing the last block of data, which is also the first sentinel block.

A second sentinel block must be written on T_p. A sentinel is already stored in the last word of P. A sentinel is stored in P_1^N, the key of the first item on the block, and the block is written on T_p.

T_p is now complete and can be rewound. Processing is stopped, thus completing the flow chart.

The computer cannot recognize a sentinel until the first sentinel block is in the memory. By setting up the flow chart to take advantage of the buffer system, it becomes impossible for the computer to transfer the first sentinel block from rI to the memory without initiating another read from Tj. The function of the second sentinel block is to prevent the computer from reading past the data in a search for another block to read.

MEMORY ALLOCATION

To facilitate the allocation of the memory, it is customary to store instructions by starting at the front of the memory and working back, and to store data by starting at the back and working forward. For this problem the memory allocation might be

CELLS	DATA
1940-1999	Output
1880-1939	Input

CODING

	0000	∼	∼	
	0001	120000	810000	$T_j \rightarrow rI$
	0002	321880	∼	$rI \rightarrow J$; $T_j \rightarrow rI$
②	0003	B01880	C01999	$J_i^N \rightarrow P_k^N$
	0004	F00035	BF1881	
	0005	SF1882	.50000	$J_i^C - J_i^M - J_i^L - J_i^O \rightarrow P_k^A$
	0006	F00036	SF1881	
	0007	SF1882	EF1999	
	0008	R00016	U00013	Ⓟ
	0009	R00022	U00019	Ⓙ
(1a)(1v)	0010	[∼	U00003]	
	0011	L00037	Q00030	$J_i^N : Z$

```
          0012    ∽
                            U00003
          ------------------------
▷P        0013    F01999               ⎫
                            B00014     ⎪
          0014   [G01940               ⎬     k + 1 → k
                            A-0038]    ⎭
          0015   [H00014
                            A00039]  *       k : 60
③         0016    ∽                          ▷P
                            U00009
          ------------------------
          0017    B00040               }         1 → k
                            C00014
          0018    531940                     P → T_P
                            U00016
          ------------------------
▷J        0019   [V31883
                            B00019]    ⎫
          0020    A-0041               ⎬     i + 1 → i
                            H00019     ⎭
          0021    A00042             *        i : 20
                            W31880
④         0022    ∽                          ▷J
                            U00010
          ------------------------
          0023    321880                     rI → J ; T_j → rI
                            B00043     }     1 → i
          0024    C00019
                            F00044     ⎫
          0025    BF1939               ⎪
                            L00037     ⎬     J^S: Z
          0026    ∽                    ⎪
                            Q00028     ⎭
          ------------------------
          0027    ∽
                            U00022
          0028    820000                     RWD* T_j
                            B00045     }     .1b
          0029    C00010
                            U00022
          ------------------------
          0030    F00037               }      Z → P^S
                            G01999
          0031    R00015
                            U00014
          ------------------------
          0032    531940                     P → T_P
                            G01940           Z → P_i^N
```

```
0033    531940                        P──►T_P
                    830000            RWD* T_P
0034    900000                        Stop
────────────────────────────
                    ⊂—⊃        ⎫
0035    001111                 ⎪
                    100000     ⎪
0036    ⊂—⊃                    ⎪
                    011111     ⎪
0037    ZZZZZZ                 ⎪
                    ⊂—⊃        ⎪
0038    000001                 ⎪
                    ⊂—⊃        ⎪
0039    098001                 ⎪
                    ⊂—⊃        ⎪
0040    G01940                 ⎬      constants
                    A-0038     ⎪
0041    000003                 ⎪
                    ⊂—⊃        ⎪
0042    068057                 ⎪
                    ⊂—⊃        ⎪
0043    V31883                 ⎪
                    B00019     ⎪
0044    111111                 ⎪
                    ⊂—⊃        ⎪
0045    F00044                 ⎪
                    BF1880     ⎭
```

Coding the resetting of an item counter consists of resetting the variable line in the item advance routine to its initial state, as shown in cells 0017, 0023and 0024.

To store a sentinel in the key of the item immediately following the last data item, the following coding technique is used. The address of the key is specified by the address part of the GOm instruction in cell 0014. The sentinel is transferred to rF by the FOm instruction in cell 0030. The UOm instruction in cell 0031 transfers control to the GOm instruction, which transfers the sentinel to the proper key. The ROm instruction in cell 0031 guarantees that, after the GOm instruction has been executed, control returns to cell 0032 to complete the ending routine.

STUDENT EXERCISES

1. A tape contains a series of two word consumption items of form

NNNNNNNNNNNN
000000CCCCCC$_\wedge$

where N - meter number
C - amount

There is at least one full block of data on the tape. Print the body of the following table.

RANGE	CONSUMPTION	METERS
1 - 100		
101 - 500		
501 - 1000		
1001 or over		

2. A tape contains a series of ten word inventory items of form

NNNNNNNNNNNN
000000QQQQQQ$_\wedge$
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX

where N - stock number
Q - quantity
X - other data

Another tape contains a series of two word items of form

NNNNNNNNNNNN
000000AAAAAA$_\wedge$

where N - stock number
A - quantity

The first item on the inventory and sales tapes have the same stock number; the second item on the tapes have the same number; and so on. There is at least one full block of data on each tape. Write the updated inventory.

SUMMARY

Magnetic tape: may be metal or mylar plastic, about .002 inches thick, one half inch wide, and up to 2400 feet in length.

Information: is written on tape in blocks of 720 characters or 60 words.

Buffers: rI and rO, one block buffers, allow for simultaneous reading, writing, processing, and rewinding.

Sentinels: following the data on tape are sentinel words consisting of six Z's in the six most significant digit positions of the words. One sentinel is placed in the key word following the last data item and another in the last word of the block. A second sentinel block has a sentinel in the first keyword of the block and another in the last word of the block.

INSTRUCTIONS

1n0000:	Read forward: Tn → rI
2n0000:	Read backward: rI ← Tn
30m:	(rI) → m
3nm:	Forward Continuous Read: (rI) → m; Tn → rI
40m:	(rI) → m
4nm:	Backward Continuous Read: (rI) → m; rI ← Tn
5nm:	60 words → Tn, 250 characters/in.
6n0000:	Rewind Tn
7nm:	60 words → Tn, 50 characters/in.
8n0000:	Rewind Tn; set interlock

Protection: is given to data and instructions by placing snap rings in the tape reels. These rings prevent recording but do not affect reading or rewinding. All recorded information can be protected by using the 8n0000 order when rewinding. The interlock prevents further use of the Uniservo until the tape is changed.

chapter 10

Timing and Efficient Use of Buffers

Generally a computer data processing application involves more than one input. For example, an inventory application involves, at least, an inventory tape and a sales tape. To use the computer in such an application, the computer must maintain, in its memory, items from both the inventory and sales tapes. Moreover, for computer efficiency both the reading of a block from the inventory tape and the reading of a block from the sales tape must be bufferred. Use of multiple buffers, one buffer for the inventory tape and another for the sales tape, is one solution to this problem. However, a buffer is an expensive piece of hardware, and the provision of multiple buffers would increase the computer's cost significantly. Thus, a technique must be found which will funnel the data through one buffer, **rI**, without sacrificing processing time.

PRESELECTION

The programming principle of preselection is one solution to the problem of buffering multiple inputs. Consider the following.

ILLUSTRATIVE EXAMPLE

A tape contains a series of ten word inventory items of form

```
NNNNNNNNNNNN
0QQQQQ‸DDDDDD
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
```

Where N - stock number
Q - quantity
D - description
X - other data

Another tape contains a series of two word sales items of form

```
NNNNNNNNNNNN
0AAAAA‸DDDDDD
```

Where N - stock number
A - quantity
D - description

The items are in ascending order by stock number on both tapes. There is at least one full block of data on each tape. Write on updated inventory.

SERVO ALLOCATION

2 - Inventory
3 - Sales
4 - Updated Inventory

FLOW CHART

Once a block of inventory items and a block of sales items have been read in the memory, the processing can begin. But before beginning the processing, the read into rI of the next block of data to be required by the computer should be initiated. The question is - Will the computer next need a block of inventory items or a block of sales items?

The example places no restriction on the nature of the stock numbers of the items. Thus,

1. There may be inventory items to which no sales items refer; that is, there may be inventory items whose stock numbers are not the same as the stock number of any sales item;

and 2. There may be more than one sales item referring to the same inventory item.

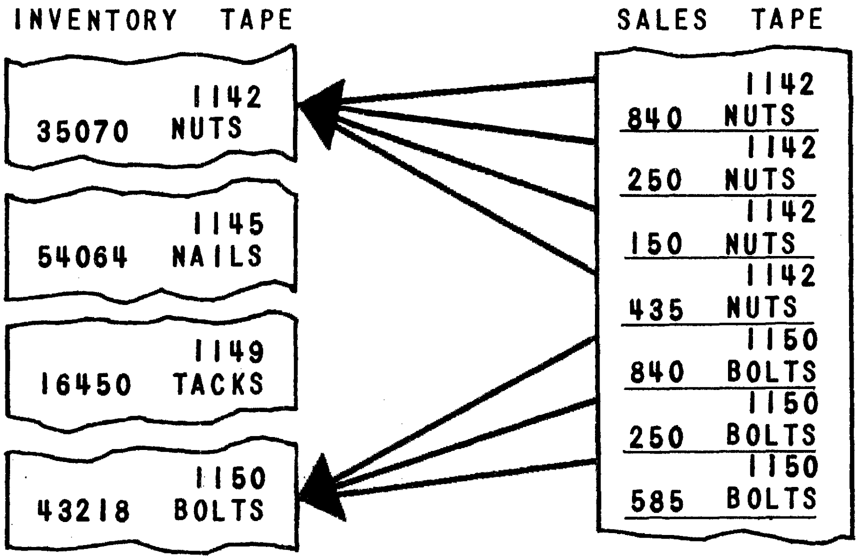


FIGURE 10-1

If all the sales items in the memory refer to inventory items in the memory, there may be more sales items not yet read into the memory which refer to the current block of inventory items. Thus, the computer will next need another block of sales items. For example,

INVENTORY ITEM STOCK NUMBERS	SALES ITEM STOCK NUMBERS
1142	1142
1145	1142
1149	1142
1150	.
1153	.
1154	.
	1153

or

INVENTORY ITEM STOCK NUMBERS	SALES ITEM STOCK NUMBERS
1142	1142
1145	1142
1149	1142
1150	.
1153	.
1154	.
	1154

If some of the sales items in the memory refer to inventory items that have not yet been read into the memory, the current block of inventory items will be processed and written before the current block of sales items is exhausted. Thus, the computer will next need another block of inventory items. For example,

INVENTORY ITEM STOCK NUMBERS	SALES ITEM STOCK NUMBERS
1142	1142
1145	1142
1149	1142
1150	.
1153	.
1154	.
	1165

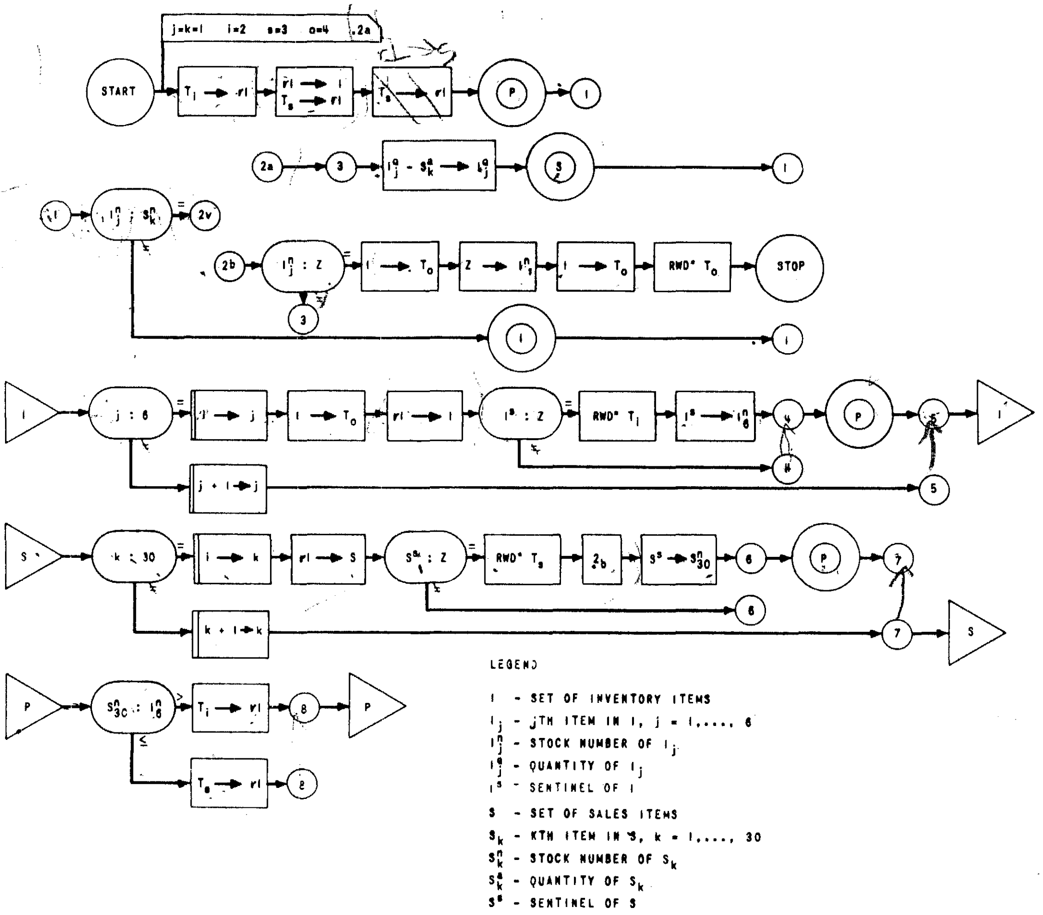
From the above, it is apparent that

1. When the stock number of the last sales item in the memory is less than or

equal to the stock number of the last inventory item in the memory, the computer will next need another block of sales items.

2. When the stock number of the last sales item is greater than the stock number of the last inventory item, the computer will next need a block of inventory items.

Based on this fact, a test for relative magnitude between the stock numbers of the last sales and inventory items permits the initiation of the read into rI of the next block of data to be required by the computer. Since the tape from which the read is to be initiated is selected before the items in the memory are processed, this programming principle is called preselection, which the following flow chart incorporates in subroutine P.



LEGEND

I - SET OF INVENTORY ITEMS
I_j - jTH ITEM IN I, j = 1,...., 6
I^n_j - STOCK NUMBER OF I_j
I^q_j - QUANTITY OF I_j
I^s - SENTINEL OF I
S - SET OF SALES ITEMS
S_k - KTH ITEM IN S, k = 1,...., 30
S^n_k - STOCK NUMBER OF S_k
S^q_k - QUANTITY OF S_k
S^s - SENTINEL OF S

FIGURE 10-2

As shown in the flow chart, when a block of items in the memory is exhausted, the only operation necessary to get the next block of items into the memory is to transfer the block from rI, since the preselection subroutine has already read the block into rI from the proper tape. Control must then go to the preselection subroutine to again determine from which tape rI is to be filled.

When a sentinel is discovered in the last word of a block, the sentinel is transferred to the key of the last item in the block to assure the proper operation of the preselection subroutine.

MEMORY ALLOCATION

1820 - 1879 - Sales Area
1880 - 1939 - Inventory Input Area
1940 - 1999 - Inventory Output Area

CODING

```
         0000   ~~~~
                           ~~~~
         0001   120000                     Ti ──► rI
                           810000
         0002   F00049
                           331880          rI ──► I ; Ts ──► rI
         0003   301820                     rI ──► S
                           ~~~~
         0004   R00041              }      (P)
                           U00038   }
        ─────────────────────────
  (1)    0005   B01880              |
                           L01820   |      I_j^N : S_k^N
         0006   ~~~~                |
                           Q00009   |
        ─────────────────────────
         0007   R00017              }      (I)
                           U00013   }
        ─────────────────────────
         0008   ~~~~
                           U00005
        ─────────────────────────
(2a)(2v) 0009  [BF1881           ]  |
                           ~~~~  ]  |
  (3)    0010   SF1821              |      I_j^Q - S_k^A ──► I_j^Q
                           EF1881   |
         0011   R00029              }      (S)
                           U00026   }
        ─────────────────────────
```

```
      0012    ~
                        U00005
      ------------------------
▷I    0013    Y11880                    
                        B00014  ┐   ⎫
      0014   [Z11940            │   ⎪
                        Y11890  ┘   ⎬    j + 1 → j
      0015   [A-0050            ┐   ⎪
                        H00014  │   ⎭
      0016    A00051            ┘        *j : 6
                        Z11880
⑤     0017    ~
                        U00008           ▷I
      ------------------------
      0018    B00052            }        1 → j
                        C00014
      0019    541940                     I → T_O
                        301880           rI → I
      0020    BF1939
                        L00053      ⎫
      0021    ~                     ⎬    I^S : Z
                        Q00024      ⎭
      ------------------------
④     0022    R00041            }        ◎P
                        U00038
      ------------------------
      0023    ~
                        U00017
      ------------------------
      0024    820000                     RWD* T_i
                        C01930           I^S → I^N_6
      0025    ~
                        U00022
      ------------------------
▷S    0026   [V21822            ┐
                        B00026  ┘   ⎫
      0027    A-0054                ⎬    k + 1 → k
                        H00026      ⎭
      0028    A00055                     *k : 30
                        W21820
⑦     0029    ~
                        U00012           ▷S
      ------------------------
      0030    B00056            }        1 → k
                        C00026
      0031    301820                     rI → S
                        BF1879      ⎫
      0032    L00053                ⎬    S^S : Z
                        Q00035      ⎭
      ------------------------
⑥     0033    R00041            }        ◎P
                        U00038
      ------------------------
```

	0034	~		
			U00029	
	0035	830000		RWD* T_s
			C01878	$S^S \rightarrow S^N_{30}$
	0036	B00057		$.2_b$
			C00009	
	0037	~		
			U00033	
▷P	0038	B01878		
			L01930	$S^N_{30} : I^N_6$
	0039	~		
			T00042	
	0040	130000		$T_s \rightarrow rI$
			~	
⑧	0041	[~		
			UOVARI]	▷P
	0042	120000		$T_i \rightarrow rI$
			U00041	
(2b)	0043	L00053		I^N_j : Z
			Q00045	
	0044	BF1881		
			U00010	
	0045	R00015		
			U00013	
	0046	I01999		
			541940	$I \rightarrow T_O$
	0047	I01940		$Z \rightarrow I^N_1$
			541940	$I \rightarrow T_O$
	0048	840000		RWD* T_O
			900000	Stop
	0049	111111		
			~	
	0050	000010		
			000010	
	0051	088000		
			~	
	0052	Z11940		
			Y11890	
	0053	ZZZZZZ		Constants
			~	
	0054	000002		
			~	
	0055	078118		
			~	
	0056	V21822		
			B00026	
	0057	BF1880		
			U00043	

STUDENT EXERCISE

A tape contains a series of twenty word policy items, each item having a policy number of form

NNNNNNNNNNNN

in the zero word. No two policy items have the same policy number. Another tape contains a series of one word policy number items of form

NNNNNNNNNNNN

No two policy number items are the same. The items are in ascending order by policy number on both tapes. There is at least one full block of data on each tape. Write a tape containing the policy items for which there is a policy number item on the policy number tape.

ROUTINE TIMING

The basic unit in the timing of routines is the minor cycle (mc), which is the time required for a Univac Computer word to pass a given point in the computer. For example, the time needed to transfer a word from memory to rA is one mc. The duration of a mc is 40 microseconds (μs).

The time needed to execute β Time On is one mc, since the execution involves transferring one word from memory to CR; and one word from CC, through the adder and back to CC; both of which occur simultaneously.

The time needed to execute γ and δ Time On varies with the instruction being executed. The time need to execute a TO period, βTO, γTO or δTO, is one mc.

The time needed to execute an instruction, the execution of which consists of transferring one word from one storage to another, is one mc. These instructions are B0m, BFm, C0m, E0m, F0m, G0m, H0m, I0m, J0m, K0m, L0m, LFm, R0m, U0m, and X0m. One mc is also required to execute the 00m instruction.

The time needed to execute a multiword transfer instruction depends on the number of words being transferred. n mc are needed to execute the Vnm and Wnm instructions; 10n mc, to execute the Ynm and Znm instructions.

The shift instructions are executed by shifting (rA) one digit position each mc. Thus, n mc are needed to execute the 0nm, -nm, ;nm and .nm instructions.

The conditional transfer of control instructions are executed in three steps.

1. Compare (rA) and (rL).
2. Set up to transfer control or skip.
3. Transfer control or skip.

One mc is needed to execute each step. Thus, three mc are needed to execute the Q0m and T0m instructions.

The add instructions are executed in three steps.

1. Transfer (m) to rX.
2. Set up to add.
3. Transfer the sum of (rA) and (rX) to rA.

One mc is needed to execute each step. Thus, three mc are needed to execute the AOm, AFm, SOm and SFm instructions.

The AHm and SHm instructions are executed in the same way as the add instructions except that two extra steps are taken.

4. Set up to transfer (rA) to m.
5. Transfer (rA) to m.

One mc is needed to execute each of the extra steps. Thus, five mc are needed to execute the AHm and SHm instructions.

The EFm instruction is executed in three steps.

1. Even characters of (rF) extract (m) into rA .
2. Set up to transfer (rA) to m.
3. Transfer (rA) to m.

One mc is needed to execute each step. Thus, three mc are needed to execute the EFm instruction.

For multiplication twenty mc, plus one mc for each addition required, are needed to execute the M0m, MFm, N0m, NFm, P0m and PFm instructions. For example, 42 mc are needed to execute a M0m instruction specifying a cell containing the word

044444444444

The computer does division by a series of additions and subtractions. The time needed to execute a divide instruction depends on the number of additions and subtractions needed to develop the quotient, which depends on the nature of the divisor. The only time a divide instruction is usedis when the nature of the divisor is not known. Thus, the only meaningful execution time for the D0m and DFm instructions is an average execution time, which is 86 mc.

SUMMARY

INSTRUCTION	EXECUTION TIME IN MINOR CYCLES
A0m	3
AFm	3
AHm	5
B0m	1
BFm	1
C0m	1
D0m[1]	86
DFm[1]	86
E0m	1
EFm	3
F0m	1
G0m	1
H0m	1
I0m	1
J0m	1
K0m	1
L0m	1
LFm	1
M0m[2]	20 + X
MFm[2]	20 + X
N0m[2]	20 + X
NFm[2]	20 + X
P0m[2]	20 + X
PFm[2]	20 + X
Q0m	3
R0m	1
S0m	3
SHm	5
SFm	3
T0m	3
U0m	1
Vnm	n
Wnm	n
X0m	1
Ynm	10n
Znm	10n

INSTRUCTION	EXECUTION TIME IN MINOR CYCLES
00m	1
. nm	n
; nm	n
−nm	n
0nm	n

1 Average
2 X = number of additions needed

TIMING AN INSTRUCTION PAIR

The above information allows the timing of an instruction pair. For example, the timing for the instruction pair

B01880A01881 .

would be as follows.

	EXECUTION TIME
β TO	1
β Time On	1
γ TO	1
γ Time On	1
δ TO	1
δ Time On	3
	8 mc
	x 40
	320 μs

TIMING A STRAIGHT LINE ROUTINE

Timing a routine that involves no iteration, a straight line routine, consists of timing each instruction pair in the routine and summing the times. For example, the timing for the processing part of the routine on page 44 would be as follows.

CODING	EXECUTION TIME
0001 B01880 A01881	8
0002 S01882 C01883	8
	16 mc
	x 40
	640 μs

TIMING AN ITERATIVE ROUTINE

Timing an iterative routine consists of timing the instruction pairs, multiplying each of these times by the number of times the associated instruction pair is executed and summing the products. For example, the timing for the processing part of the routine on page 98 would be as follows.

CODING	EXECUTION TIME	NO. OF TIMES EXECUTED	TOTAL TIME
0001 B00007 AH0003	10	31	310
0002 L00008 Q00005	8	31	248
0003 B01878 A01879	8	30	240
0004 AH0006 U00001	10	30	310
			1098 mc
			x 40
			43,920 μs
			x .001
			43.92 ms*

* milliseconds

TIMING A BRANCH ROUTINE

A branch routine is an iterative routine in which an item can be processed in one of many ways depending on the nature of the item. For example, the routine on page136is a branch routine, theprocessing of a job item being dependent on whether salesman A or salesman B closed the contract, and also on whether or not the job netted more than $250. To determine the number of times each instruction pair in a branch routine is executed it is necessary to know

1. How many contracts salesman A closed
2. How many contracts salesman B closed
3. How many of the jobs contracted for by salesman A netted more than $250,

and 4. How many of the jobs contracted for by salesman B netted more than $250?

Usually this information will not be available for the specific 30 job items to be processed, but statistics may be available for jobs in general. For example, it may be known that, in general, salesman A closes about as many contracts as salesman B, and that one out of every five jobs contracted for by salesman A nets more than $250, while one out of every three contracted for by salesman B nets more than $250. On the basis of such statistics a branch routine can be timed. Using the above statistics, the timing would be as follows.

CODING	EXECUTION TIME	NO.OF TIMES EXECUTED	TOTAL TIME
0001 F00029 BF1880	6	30	180
0002 L00030 Q00008	8	30	240
0003 B00026 F00031	6	15	90
0004 AF1880 C00026	8	15	120
0005 R00021 U00013	6	15	90
0006 B00028 A00032	8	5	40
0007 C00028 U00017	6	5	30
0008 B00025 F00031	6	15	90
0009 AF1880 C00025	8	15	120
0010 R00021 U00013	6	15	90
0011 B00027 A00032	8	3	24
0012 C00027 U00017	6	3	18
0013 BF1880 SF1881	8	30	240
0014 , 50000 F00033	10	30	300
0015 SF1880 SF1881	10	30	300
0016 L00034 T00021	8	30	240
0017 B00019 L00035	6	30	180
0018 A00036 Q00022	10	30	300
0019 V21882 W21880	8	29	232
0020 C00019 U00001	6	29	174
0021 ␣ U0VARI	6	8	48
			3146 mc
			x 40
			125,840 μs
			x .001
			125.84 ms

TAPE INSTRUCTIONS

There is associated with the execution of a tape instruction an execution time, that is, a characteristic duration of time during which the computer is concerned with the execution of the instruction. However, also associated with the execution of a tape instruction is a tape time, a characteristic duration of time during which the specified Uniservo is concerned with the execution of the instruction. The execution time of a tape instruction is included in the tape time, which is 43.8 ms for 250 character per inch density.

At the end of the execution time of a tape instruction, the computer is freed to continue processing. However, if, during the tape time of the tape instruction, the computer is instructed to execute an input-output instruction that either

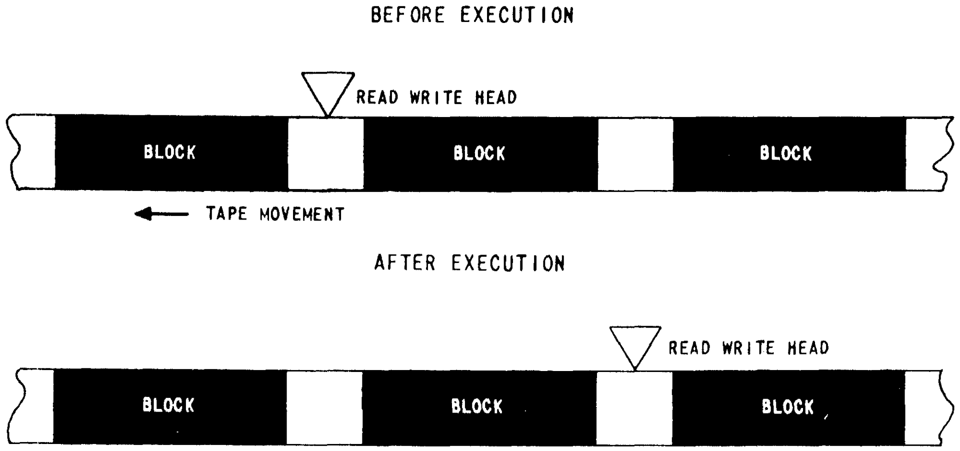
1. is of the same kind as this tape order (there are two kinds of input-output instructions, read instructions and write instructions)

or 2. specifies the same Uniservo as this tape instruction,

the computer will wait until the end of this tape time before continuing to execute instructions. During the waiting time the computer is said to be interlocked.

Thus, if the computer initiates the execution of a 1nm instruction, 43.8 ms will pass before the computer can execute a read instruction or a tape instruction specifying Uniservo n. At the end of the execution time of the 1nm instruction and during the remainder of the 43.8 ms, the computer will execute any other combination of instructions except two write instructions, since 43.8 ms must also pass between the execution of two write instructions.

At the beginning of the execution of a tape instruction the read-write head of the specified Uniservo is located at the space between blocks, halfway between the block to the right and the block to the left. During the execution of the tape instruction the read head passes over the rest of this space between blocks; over the block being read or written; and at the end of the execution of the instruction, will once more be located at a space between blocks, halfway between blocks.


BEFORE EXECUTION

AFTER EXECUTION

FIGURE 10-3

No reading or writing is done while the read-write head is passing over the space between blocks. This passage is used to accelerate the tape before the read or write operation and to decelerate the tape after the read or write operation. That part of tape time during which a read or write operation is occurring is known as read-write time; that part during which the read-write head is passing over the space between blocks is known as start-stop time. A Uniservo is being developed that will operate as follows. If the computer is interlocked during any part of the time between the execution of two consecutive tape instructions that specify the same Uniservo and move the tape in the same direction, the tape will not be stopped and restarted between the execution of the instructions. Instead, the tape will continue to move at 100" per sec. from the initiation of the first instruction to the end of the execution of the second. Such operation will reduce start-stop time, and consequently, tape time, by 5 ms.

EXECUTION TIME

The execution and tape times of all input-output instructions are as follows:

INSTRUCTION	EXECUTION TIME	TAPE TIME
1nm	2500 μs	43.8 ms
2nm	2500 μs	43.8 ms
30m	2680 μs	
3nm	2680 μs	43.8 ms
40m	2680 μs	
4nm	2680 μs	43.8 ms
5nm	2680 μs	43.8 ms
7nm	2680 μs	43.8 ms

TIMING A ROUTINE INVOLVING TAPE

The timing of a routine involving tape must take into consideration, not only processing time, but also the possibility of interlock, since both affect running time. Interlock is taken into consideration in the following way.

The running of a routine involving tape can be considered a repetition of a processing circle. For example, the circle of the routine on page 183 – 185 is as follows.

1. A block of items is read from the job tape to rI (the 321880 instruction in cell 0023).

2. The block of job items is processed (the instructions in cells 0003-0010, 0013-0016 and 0019-0027).

3. (rI) are transferred to the input area (the 321880 instruction in cell 0023).

4. Steps 1 - 3 are repeated.

5. Steps 1 and 2 are repeated.

6. The output item counter is reset (the instructions in cell 0017).

7. The block of profit items is written (the 531940 instruction in cell 0018).

8. The end of block coding for profit items is executed (the instructions in cells 0009, 0016, 0018 - 0021).

9. (rI) are transferred to the input area (the 321880 instruction in cell 0023).

The circle is complete, and processing returns to step 1.

The timing of a routine involving tape consists of timing the circle, which is done as follows. Within the circle will be certain input-output instructions the combination of which may cause interlock. These instructions divide the circle into portions. Each portion is timed, and the sum of these times will be the circle time of the routine.

A portion is timed as follows. The processing time for the portion is calculated. If this time is 43.8 ms or more, the computer will not be interlocked, and running time for the portion will be processing time; otherwise, the computer will be interlocked, and running time will be tape time, 43.8 ms.

In the above circle the portions are as follows.

A. Between steps 1 and 3.

B. Between steps 1 and 9 after the second repetition of step 1 in the circle.

The following is the timing for portion A.

CODING			EXECUTION TIME	NO.OF TIMES EXECUTED	TOTAL TIME
0000	⌴	⌴	6mc	1	6mc
0003	B01880	C01999	6mc	20	120mc
0004	F00035	BF1881	6mc	20	120mc
0005	SF1882	.50000	12mc	20	240mc
0006	F00036	SF1881	8mc	20	160mc
0007	SF1882	EF1999	10mc	20	200mc
0008	R00016	U00013	6mc	20	120mc
0009	R00022	U00019	6mc	20	120mc
0010	⌴	U00003	6mc	20	120mc
0013	F01999	B00014	6mc	20	120mc
0014	G01940	A-0038	8mc	20	160mc
0015	H00014	A00039	8mc	20	160mc
0016	⌴	U00009	6mc	20	120mc
0019	V31883	B00019	8mc	20	160mc
0020	A-0041	H00019	8mc	20	160mc
0021	A00042	W31880	10mc	20	200mc
0022	⌴	U00010	6mc	20	120mc
0023	321880	B00043	5mc+2680 μs	1	5mc+2680 μs
0024	C00019	F00044	6mc	1	6mc
0025	BF1939	L00037	6mc	1	6mc
0026	⌴	Q00028	8mc	1	8mc
0027	⌴	U00022	6mc	1	6mc
					2437mc+2680 μs
					x 40
					97,480
					+ 2680
					100,160 μs
					x .001
					100.16ms

100.16 ms are greater than 43.8 ms. Thus, running time for portion A is processing time, 100.16 ms.

The following is the timing for portion B,

CODING			EXECUTION TIME	NO.OF TIMES EXECUTED	TOTAL TIME
0000	↪	↪	6mc	2	12mc
0003	B01880	C01999	6mc	20	120mc
0004	F00035	BF1881	6mc	20	120mc
0005	SF1882	.50000	12mc	20	240mc
0006	F00036	SF1881	8mc	20	160mc
0007	SF1882	EF1999	10mc	20	200mc
0008	R00016	U00013	6mc	20	120mc
0009	R00022	U00019	6mc	20	120mc
0010	↪	U00003	6mc	20	120mc
0013	F01999	B00014	6mc	20	120mc
0014	G01940	A-0038	8mc	20	160mc
0015	H00014	A00039	8mc	20	160mc
0016	↪	U00009	6mc	20	120mc
0017	B00040	C00014	6mc	1	6mc
0018	531940	U00016	5mc+2680μs	1	5mc+2680μs
0019	V31883	B00019	8mc	20	160mc
0020	A-0041	H00019	8mc	20	160mc
0021	A00042	W31880	10mc	20	200mc
0022	↪	U00010	6mc	20	120mc
0023	321880	B00043	5mc+2680μs	1	5mc+2680μs
0024	C00019	F00044	6mc	1	6mc
0025	BF1939	L00037	6mc	1	6mc
0026	↪	Q00028	8mc	1	8mc
0027	↪	U00022	6mc	1	6mc
					2454mc+5360μs
					x 40
					98,160
					+ 5,360
					103,520μs
					x .001
					103.52 ms

Thus, running time for portion B is processing time, 103.52 ms.

In one circle portion A is executed twice, and portion B, once.

Thus, circle time is

2 (Portion A Time) =	2(100.16) =	200.32
Portion B Time =		103.52
Circle Time		303.84 ms

In one circle three blocks of job items are processed. Dividing circle time by three will give an average block time.

$$303.84 \div 3 = 101.28\text{ms}$$

Running time for a routine involving tape always depends on the number of blocks to be processed. For example, if the job tape contained 4500 blocks of data, running time would be calculated as follows.

$$\begin{array}{r} 101.28 \\ \underline{\times \;\; 4500} \\ 455{,}780 \text{ ms} \\ \underline{\times \; .001} \\ 455.78 \text{ sec.} \end{array}$$

Dividing 455.78 by 60 gives a running time of 7.6 minutes.

In the above time estimate, time needed to execute the beginning and ending subroutines was ignored. This procedure is customary, since this time is trivial.

GRAPHICAL REPRESENTATION

The running time of a routine may be represented graphically. For example, the following is the graphical representation of the running time just computed.

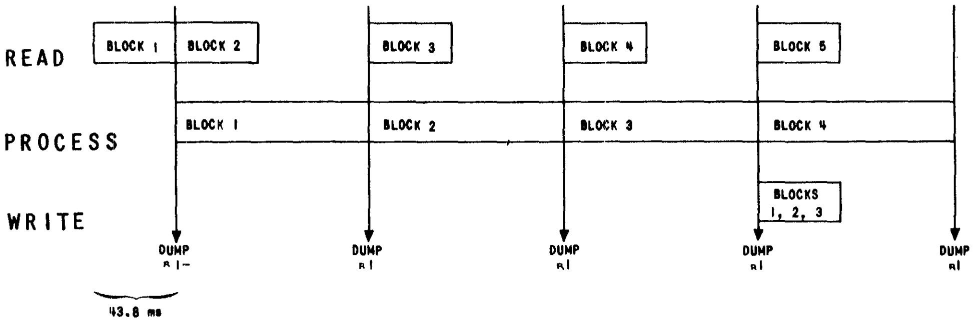


FIGURE 10.4

Graphical representation presents a concise picture that can be easily inspected to determine whether or not the most efficient use of rI is being made.

TIMING A BRANCH ROUTINE INVOLVING TAPE

Consider the routine on page 193. To time this routine, it is necessary to know the ratio of inventory items to sales items, called the activity of the inventory file. Suppose the activity were 16.7%, that is, for every six inventory items there is, in general, one sales item. The circle would then be as follows.

1. A block of inventory items is written (the 541940 instruction in cell 0019).

2. A block of inventory items, and one sales item are processed (the instructions in cells 0005-0019, 0023, 0026-0029, 0041 and 0042).

3. Steps 1 and 2 are repeated 28 times.

4. A block of inventory items is written (the 541940 instruction in cell 0019).

5. Control goes to cell 0040 (the instructions in cells 0019-0022, 0038, and 0039).

6. A block of items is read from the sales tape to rI (the 130000 instruction in cell 0040).

7. A sales item is processed (the instructions in cells 0005, 0006, 0008-0011, 0017, 0023, 0026-0028, 0030, 0040 and 0041).

8. (rI) are transferred to the sales input area (the 301820 instruction in cell 0031).

9. A block of inventory items is processed (the instructions in cells 0005-0008, 0012-0018, 0029, 0031-0034, 0038, 0039, 0041 and 0042).

The circle is complete, and processing returns to step 1. The portions are as follows.

A. Between step 1 and the repetition of step 1.

B. Between steps 6 and 8.

In addition to the time for the portions, the time for steps 5 and 9 must be included in circle time, since they are not included in either portion. The following is the timing for portion A.

CODING			EXECUTION TIME	NO.OF TIMES EXECUTED	TOTAL TIME
0000	└─┐	└─┐	6mc	1	6mc
0005	B01880	L01820	6mc	6	36mc
0006	└─┐	Q00009	8mc	6	48mc
0007	R00017	U00013	6mc	6	36mc
0008	└─┐	U00005	6mc	6	36mc
0009	BF1881	└─┐	6mc	1	6mc
0010	SF1821	EF1881	10mc	1	10mc
0011	R00029	U00026	6mc	1	6mc
0012	└─┐	U00005	6mc	1	6mc
0013	Y11880	B00014	15mc	6	90mc
0014	Z11940	Y11890	24mc	6	144mc
0015	A-0050	H00014	8mc	6	48mc
0016	A00051	Z11880	17mc	6	102mc
0017	└─┐	U00008	6mc	6	36mc
0018	B00052	C00014	6mc	1	6mc
0019	541940	301880	4mc + 5360 μs	1	4mc + 5360 μs
0020	BF1939	L00053	6mc	1	6mc
0021	└─┐	Q00024	8mc	1	8mc
0022	R00041	U00038	6mc	1	6mc
0023	└─┐	U00017	6mc	1	6mc
0026	V21822	B00026	7mc	1	7mc
0027	A-0054	H00026	8mc	1	8mc
0028	A00055	W21820	9mc	1	9mc
0029	└─┐	U00012	6mc	1	6mc
0038	B01878	L01930	6mc	1	6mc
0039	└─┐	T00042	8mc	1	8mc
0041	└─┐	U0VARI	6mc	1	6mc
0042	120000	U00041	5mc + 2500 μs	1	5mc + 2500 μs
					701mc + 7860 μs
					x40
					28,040
					+ 7860
					35,900 μs
					x .001
					35.9 ms

35.9 ms are not greater than 43.8 ms. Thus, running time for portion A is tape time, 43.8 ms.

The following is the timing for portion B.

CODING			EXECUTION TIME	NO.OF TIMES EXECUTED	TOTAL TIME
0000	└──┐	└──┐	6mc	1	6mc
0005	B01880	L01820	6mc	1	6mc
0006	└──┐	Q00009	8mc	1	8mc
0008	└──┐	U00005	6mc	1	6mc
0009	BF1881	└──┐	6mc	1	6mc
0010	SF1821	EF1881	10mc	1	10mc
0011	R00029	U00026	6mc	1	6mc
0017	└──┐	U00008	6mc	1	6mc
0023	└──┐	U00017	6mc	1	6mc
0026	V21822	B00026	7mc	1	7mc
0027	A-0054	H00026	8mc	1	8mc
0028	A00055	W21820	9mc	1	9mc
0030	B00056	C00026	6mc	1	6mc
0031	301820	└──┐	3mc[1]	1	3mc
0040	130000	└──┐	2mc+2500 μs[2]	1	2mc+2500 μs
0041	└──┐	U0VARI	6mc	1	6mc
					101mc+2500 μs
					x 40
					4040
					+ 2500
					6540 μs
					x .001
					6.54 μs

1. β TO, β Time On, γ TO; the rest of this three stage cycle is part of step 9

2. γ Time On, δ TO, δ Time On; the rest of this three stage cycle is part of step 5.

Thus, running time for portion B is tape time, 43.8 μs.

The following is the timing for step 5.

CODING			EXECUTION TIME	NO.OF TIMES EXECUTED	TOTAL TIME
0019	541940	301880	1mc + 5360 μs	1	1mc + 5360 μs
0020	BF1939	L00053	6mc	1	6mc
0021	⌴	Q00024	8mc	1	8mc
0022	R00041	U00038	6mc	1	6mc
0038	B01878	L01930	6mc	1	6mc
0039	⌴	T00042	8mc	1	8mc
0040	130000	⌴	3mc	1	3mc
					38mc + 5360 μs
					x 40
					1520
					+ 5360
					6880 μ s
					x .001
					6.88 ms

The following is the timing for step 9.

CODING			EXECUTION TIME	NO.OF TIMES EXECUTED	TOTAL TIME
0000	⌴	⌴	6mc	1	6mc
0005	B01880	L01820	6mc	6	36mc
0006	⌴	Q00009	8mc	6	48mc
0007	R00017	U00013	6mc	6	36mc
0008	⌴	U00005	6mc	5	30mc
0012	⌴	U00005	6mc	1	6mc
0013	Y11880	B00014	16mc	6	96mc
0014	Z11940	Y11890	24mc	6	144mc
0015	A-0050	H00014	8mc	6	48mc
0016	A00051	Z11880	17mc	6	102mc
0017	⌴	U00008	6mc	5	30mc
0018	B00052	C00014	6mc	1	6mc
0019	541940	⌴	3mc	1	3mc
0029	⌴	U00012	6mc	1	6mc
0031	301820	BF1879	2mc + 2680 μs	1	2mc + 2680 μs
0032	L00053	Q00035	8mc	1	8mc
0033	R00041	U00038	6mc	1	6mc
0034	⌴	U00029	6mc	1	6mc
0038	B01878	L01930	6mc	1	6mc
0039	⌴	T00042	8mc	1	8mc
0041	⌴	UOVARI	6mc	1	6mc
0042	120000	U00041	5mc + 2500 μs	1	5mc + 2500 μs
					674mc + 5180 μs
					x 40
					26,960
					+ 5,180
					32,140 μs
					x .001
					32.14 ms

In one circle portion A is executed 29 times; and portion B and steps 5 and 9, once each. Thus, cycle time is

$$\begin{array}{rclr}
29(\text{Portion A Time}) & = & 29(43.8) \; = & 1270.2 \\
\text{Portion B Time} & = & & 43.8 \\
\text{Step 5 Time} & = & & 6.88 \\
\text{Step 9 Time} & = & & \underline{32.14} \\
\text{Cycle Time} & & & 1353.02 \text{ ms}
\end{array}$$

In one circle 30 blocks of inventory items are processed. Thus, average block time is

$$\frac{1353.02}{30} = 45.101 \text{ ms}$$

If the inventory tape contained 4500 blocks, running time would be

$$\begin{array}{r}
45.101 \\
\underline{\times \; 4500} \\
202{,}954.5 \text{ ms} \\
\underline{\times \; .001} \\
202.955 \text{ sec.}
\end{array}$$

or 3.38 minutes.

STANDBY BLOCK METHOD

The standby block method is another programming technique for the solution of the problem of buffering multiple inputs. While requiring more memory space than the preselection subroutine, the standby block subroutine is usually more efficient in terms of running time.

The principle of the standby block method is to allocate to each input a 60 word standby area as well as a 60 word input area. For example, for two input tapes, Ta and Tb, an input area and a standby area, A and $\bar{A}$, would be allocated to Ta; and an input area and a standby area, B and $\bar{B}$, to Tb.

Initially, the first block of items from T_a is read into area A; the first block from T_b, into area B; the second block from T_a, into $\bar{A}$; and the second block from T_b, into rI; giving the following configuration, which will be referred to as configuration 1.

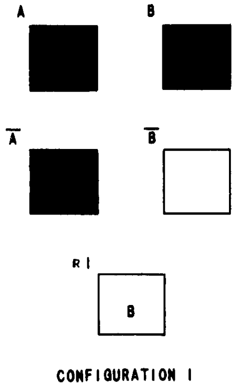


FIGURE 10-5

The following discussion of the operation of the standby block technique is based on figure 10-6.

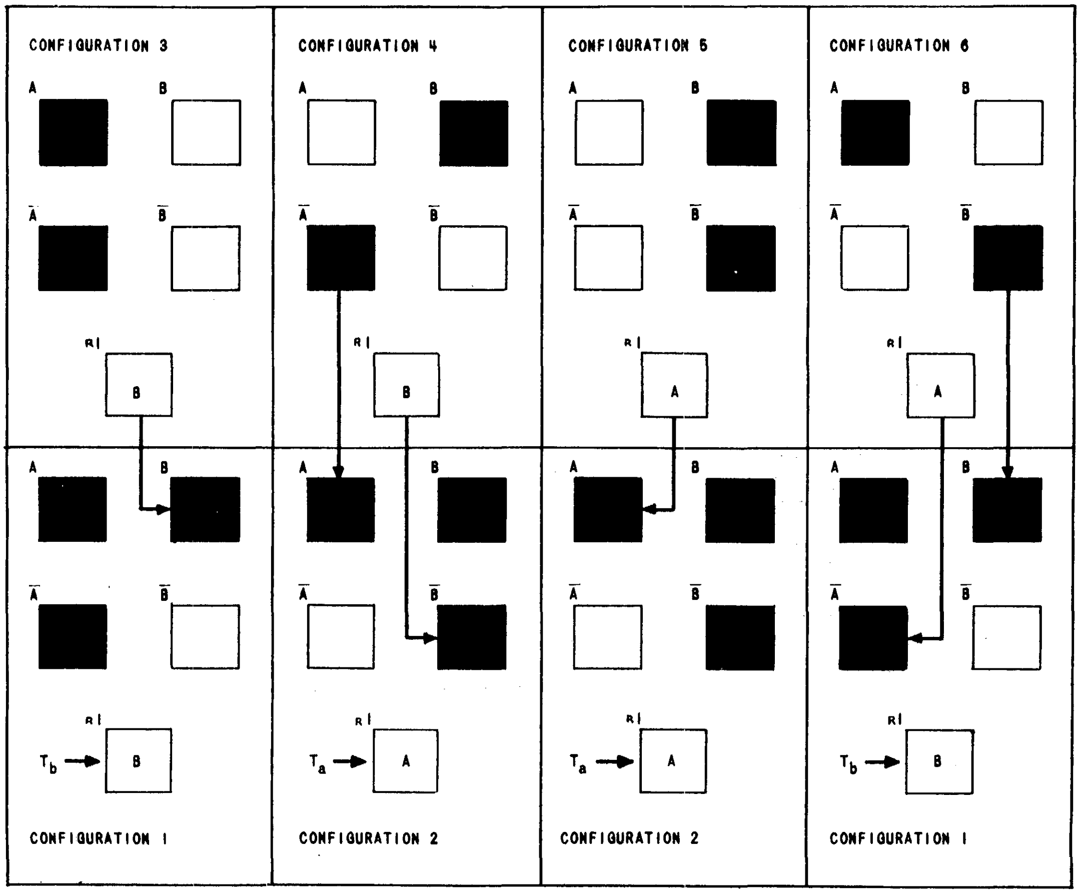


FIGURE 10-6

If in configuration 1, the B items are exhausted first (configuration 3), (rI) are transferred to area B, and a block is·read from T_b into rI, recreating configuration 1.

If, in configuration 1, the A items are exhausted (configuration 4), the contents of area $\bar{A}$ are transferred to area A, (rI) are transferred to area $\bar{B}$, and a block is read from T_a into rI, creating configuration 2.

If, in configuration 2, the A items are exhausted (configuration 5), (rI) are transferred to area A, and a block is read from T_a into rI, recreating configuration 2.

If, in configuration 2, the B items are exhausted (configuration 6), the contents of area $\bar{B}$ are transferred to area B, (rI) are transferred to area $\bar{A}$, and a block is read from T_b into rI, creating a configuration I.

Configurations 1 - 6 exhaust the possibilities. Thus, besides the block of A items and the block of B items currently being processed, there is always another block of A items and another block of B items in electronic storage, either in rI or in a standby area.

The following is an abbreviated flow chart of the standby block technique.

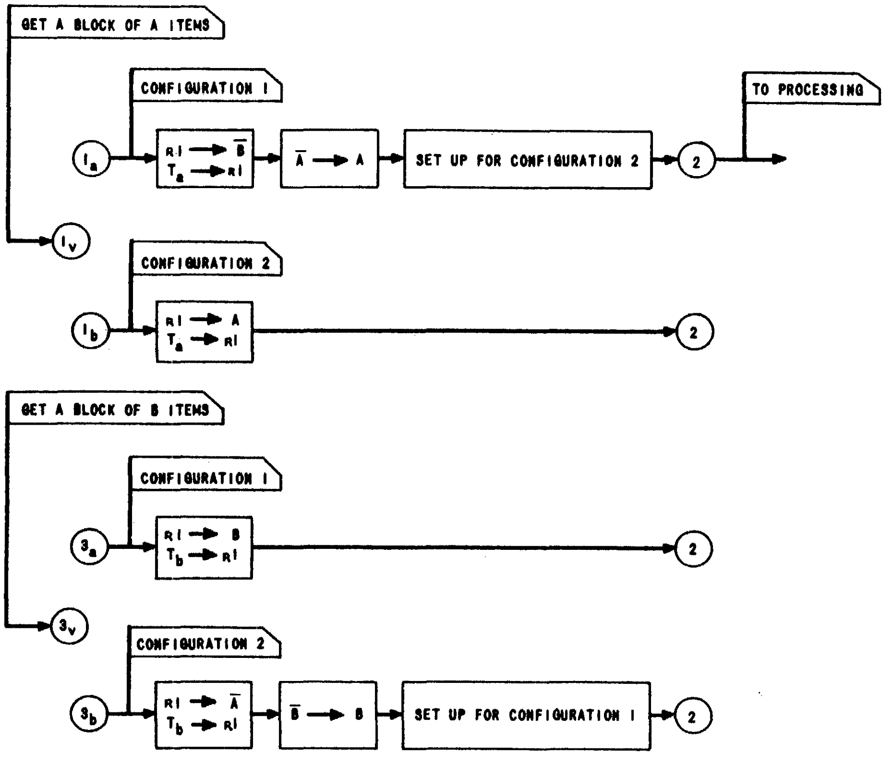


FIGURE 10-7

Basically, the reason why the standby block method is faster then the preselection technique is that it requires only one input order, a 3nm, whereas preselection requires two: a 1nm followed by a 30m. Then, because the amounts of data in input files usually differ greatly, the master file is advanced with a minimum number of instructions besides the 3nm.

STUDENT EXERCISE

Flow chart and code the standby block technique.

chapter 11

Univac Supervisory Control Panel Operations

The Supervisory Control Panel permits manual intervention into the otherwise automatic operation of the computer. There are two ways in which manual operations become of use to the programmer. First, the running of a routine - the execution of the routine by the computer - requires certain manual operations, such as clear C and initial read. Secondly, manual operations are of use in debugging.

An error in a routine - an aspect of a routine which causes the routine, when run, to produce unexpected results - is called a bug, and the process of eliminating bugs from a routine is called debugging. A programmer cannot be sure that a routine is correct - that is, has no bugs - until he has run the routine against all possible types of input and determined that the routine produces the expected output. If, in such a debugging run, a bug is detected, pertinent information about the bug can often be obtained by manual intervention into the running of the routine.

The execution of the 10m instruction is an example of a manual operation that may be required for the running of a routine.

THE 10m INSTRUCTION

INSTRUCTION	OPERATION
10m	SCK→m

Transfer the word typed on the Supervisory Control Keyboard (SCK) to m.

The 10m instruction is a two digit instruction.

The SCK is a modified typewriter keyboard located on the Supervisory Control Panel. Besides the standard typewriter keys, the SCK includes

1. keys for Univac characters not found on a typewriter keyboard,
2. a special bank of numeric keys for rapid typing of numeric information,

and 3. other keys used in the manual operation of the computer.

The computer executes the 10m instruction as follows. When the 10m instruction is transferred to SR, the computer stalls and lights a neon, called the input ready neon, on the Supervisory Control Panel, thus indicating that it is ready to accept the type in of one word on the SCK. The operator types 12 characters on the SCK and then depresses the "word release" key. The word typed is transferred to the cell specified by the 10m instruction.

One use of the 10m instruction is to allow the type in of constants which vary from one running of a routine to the next, such as the date.

CONDITIONAL TRANSFER BREAKPOINTS

There is, on the Supervisory Control Panel, a bank of 12 buttons called conditional transfer breakpoint selector buttons. Ten of the buttons are numbered 0-9, one is labelled "all", and one is labelled "release". If a number, 0-9, is coded in the second instruction digit of a conditional transfer of control instruction, the computer can be made to stop with this instruction in the SR. To cause the stoppage the conditional transfer selector button corresponding to the second instruction digit of the Qnm or Tnm must be depressed. The computer makes the comparison and indicates whether or not transfer of control will occur, stopping before the transfer is effected. If the computer is to transfer control, the conditional transfer neon on the Supervisory Control Panel will be lit; if not, the no transfer neon will be lit. If transfer of control is not indicated, the operator can cause a transfer of control by depressing a button, called "force transfer". If transfer of control is indicated, the operator can prevent transfer of control by depressing a button, called "no transfer".

One use of conditional transfer breakpoints is for manual control. A conditional transfer breakpoint can be coded at a crucial point in a routine, and when the computer reaches this point, the operator, by operating the transfer buttons, can choose the processing that the computer is to follow. For example, some routines are coded for a certain number of Uniservos but provide an option for using less. The option can be in the form of a conditional transfer breakpoint that normally does not transfer control. If the lesser number of Uniservos is to be used, the operator can depress the appropriate conditional transfer breakpoint selector button and force transfer when the computer reaches the breakpoint, thus causing the computer to follow a path other than normal.

Breakpoints are also used in bugshooting. If a bug cannot be found by desk checking, conditional transfer breakpoints can be inserted at crucial points in the routine. If the associated conditional transfer breakpoint selector buttons are depressed, the computer will stop everytime the conditional transfer instructions are set up in the SR. The contents of crucial cells and registers can then be investigated for correctness before continuing with the routine. This investigation is conducted after the computer has been set to operate on other than the continuous mode and can be made as follows: (Non-continuous operation is made possible by operating the Interrupted Operation Buttons, which will be described later).

PRINTING FROM THE SUPERVISORY CONTROL PANEL

By means of switches on the Supervisory Control Panel the operator can stop the computer, set up an instruction in SR, cause the computer to execute the instruction, and still prevent the computer from losing its place in the routine whose execution has been interrupted. Thus, if a programmer wants to know the contents of a given cell, the operator can set up a 50m instruction, with m the given cell, in SR and cause the computer to print the contents of the cell. The contents of a register can be investigated in a similar fashion, as follows.

There is, on the Supervisory Control Panel, a bank of eight buttons, called type out selector buttons and labelled M, F, L, A, X, CR, C and "empty". Only when type out selector button M is depressed will the computer execute the 50m instruction as defined. If, for example, type out selector button A was depressed when a 50m instruction was executed, the contents of, not m, but rA would be printed. Similarly, type out selector button F causes (rF) to be printed; L, (rL); X, (rX); CR, (CR); and C, (CC). Thus, if a programmer wants to know the contents of a given register, the operator can set up a 50m instruction, depress the appropriate type out selector button, and cause the computer to print the contents of the register.

Whenever printing on the SCP takes place the characters are monitored according to the position of a function switch. Some characters cause printer action, such as carriage return, tabulate, space, etc. There are times, however, when it is desired to know what the character is rather than have the action take place. When the function switch is in the Normal position action takes place whereas when the switch is in the Computer Digit position a substitute character is printed.

The character chart in the back of the manual indicates the action, or character printed, when a given character is transferred to the printer.

THE ALL CONDITIONAL TRANSFER BREAKPOINT SELECTOR BUTTON

Depressing the conditional transfer breakpoint selector button labelled "all" causes the computer to stop on all conditional transfer instructions. One use of the "all" button is in the debugging of a type of bug calleda closed loop. It is not uncommon for a routine to be coded in such a manner that a loop of instructions are created from which there is no exit. There is a characteristic noise, created by the transfer of data from one storage to another, which is amplified and emitted from a speaker behind the Supervisory Control Panel. When a closed loop is entered, the noise takes on a repetitious character. If the "all" button is then depressed, the computer will stop on the first conditional transfer instruction encountered, if there is one in the loop. Depressing a bar, called the start bar, on the SCK will cause the computer to continue executing instructions until the next conditional transfer instruction is reached. If this process is continued; and if each time the computer stops, the programmer notes

1. the location and nature of the conditional transfer of control instruction on which the computer stopped

and 2. whether or not the computer is going to transfer control;

the path or the closed loop through the coding will soon be determined. The conditional transfer of control instruction on which the computer stopped can be determined in one of two ways.

1. The operator can read (SR) from a series of neons on the Supervisory Control Panel. Thus, the operator can tell the programmer on what conditional transfer of control instruction the computer stopped, and the programmer can locate the instruction in his copy of the coding.

2. (CC) can be printed. The address printed will be one more than the address of the cell in which the conditional transfer of control instruction is stored.

If the closed loop does not contain any conditional transfer of control instructions, the path of the closed loop can be determined by executing the instructions in the loop one at a time.

INTERRUPTED OPERATION

Interrupted Operation is controlled by a bank of five buttons on the Supervisory Control Panel. The buttons are labelled one addition, one step, one operation, one instruction and continuous. Of these, only the continuous and one instruction buttons are of significance here.

If the continuous button is depressed, the computer is said to be "on continuous" and operates in the following manner. When the start bar is depressed, the computer starts executing instructions and will not stop until either a 90m instruction is executed or a breakpoint is reached. Once the computer stops, it will not start again until the bar is depressed. However, if the computer is placed in the one instruction mode by depressing the one instruction button and the start bar is then depressed, the computer will stop during a TO period, βTO, γTO or δTO. Thus, if a closed loop contains no conditional transfer of control instructions, the operator can place the computer on one instruction, and the computer will stop on the first TO period encountered. Depressing the start bar will cause the computer to complete the execution of the current stage of the three stage cycle and stop during the next TO period. If this process is continued; and if each time the computer stops on γ or δ Time, the programmer notes the location and nature of the instruction just transferred to SR; the path of the closed loop, and possibly the reason for it, will soon be determined.

THE RELEASE CONDITIONAL TRANSFER BREAKPOINT SELECTOR BUTTON

With the exception of the conditional transfer breakpoint selector button labelled "release", the conditional transfer breakpoint selector buttons are such that, when depressed, they remain depressed. The depression of the "release" button releases all of the buttons.

OTHER BREAKPOINTS

There are breakpoints other than conditional transfer breakpoints. One is the comma breakpoint. If a comma is coded in the first instruction digit of an instruction, and if a switch, called the comma breakpoint switch, on the Supervisory Control Panel is locked in the down position, the computer will stop when the ,0m instruction is transferred to SR. If the comma breakpoint switch is in the normal position, the computer interprets a ,0m instruction as a skip.

A third breakpoint is the fifty breakpoint. If a switch, called the type out breakpoint switch, on the Supervisory Control Panel is locked in the down position, every time a 50m instruction is transferred to SR the computer will stop before printing. If the type out breakpoint switch is put in the center position, the normal position, the computer interprets 50m instructions as defined. The switch can also be locked in the up position, called the skip type out position, which causes the computer to interpret all 50m instructions as skips. The skip type out position of the type out breakpoint switch allows the programmer to speed up the execution of a routine by skipping .type outs that otherwise would normally occur.

MANUAL ALTERATION OF INSTRUCTIONS IN THE MEMORY

It often happens that, in a debugging run, the computer will stall, or "hang up", on a bug, and after a short investigation the programmer decides that, by a slight alteration of the instructions, the bug can be eliminated. Rather than preparing a new instruction tape to test his theory, the programmer can make the alterations in the memory by the following manual operations.

The instruction tape is initial read. By placing the computer on one instruction, the operator can then step the computer, stage by stage, through the instructions that read the rest of the instructions into the memory. At this point the operator can set up, in SR, 10m instructions to the cells the contents of which the programmer wants to modify. The execution of the 10m instructions completes the modification, and the corrected routine can then be run by putting the computer on continuous. It is normal operating procedure to first print out the words to be altered.

THE FILL OPERATION

If the programmer wants to modify the contents of a series of consecutive cells, he can use a procedure, called the fill operation, that is faster than the setting up of 10m instruction in SR. By operation of the fill memory switch, the operator can cause the computer to set up in SR a 10m instruction to the cell specified by the four least significant digits of CC. After this 10m instruction has been executed, the computer automatically increases (CC) by one and once more sets up a 10m instruction to the cell specified. This process can be continued for the contents of as many cells as the programmer wants to modify.

If the programmer wants to start the fill operation with cell 0000, a word of zeros can be transferred to CC by depressing a button, called the clear C button. Depression of the clear C button is the operation referred to as "clear C". If the programmer wants to start the fill operation with some cell other than cell 0000, the proper address can be transferred to CC by the SCICR operation.

SCICR

By operation at the Supervisory Control Panel, the operator can perform the operation known as SCICR (Supervisory Control Input to CR). This operation allows the operator to type 12 characters on SCK and have the resulting word transferred to CR. If, for example, the programmer wanted to start a fill operation at cell 0029, the operator could SCICR a 0 0m U0m instruction pair. The U0m instruction would specify cell 0029. Then, by putting the computer on one instruction, the operator could cause the computer to execute the 00mU0m instruction pair. At the end of the execution the address in CC would be 0029. The operator can then begin the fill operation at cell 0029.

GENERATING DATA

To debug a routine, data must first be provided for the routine. Knowledge of the nature of the data aids materially in locating bugs. Thus, initial data is usually prepared by the programmer. In many cases it is not necessary for the programmer to write out such data and have the data unityped. Instead, a rather simple routine can be coded that, when executed, generates the data as its output. The correctness of such a generator routine can be checked visually by printing the output on the Univac High-Speed Printer.

DEBUGGING PROCEDURE

When the programmer takes his routine on the computer for a debugging run, he should have with him all information pertinent to the routine, and always a copy of the flow chart and coding. Usual debugging procedure is to run the routine for the first time with the computer on continuous. The routine may hang up on a bug, enter a closed loop or run to completion. When the computer encounters a bug, the programmer must note all pertinent information about the bug, preferably by writing it down. For example, if the routine were to hang up on an adder-alphabetic error, the pertinent information would be the answers to the questions:

1. How long after the execution of the routine started did the routine hang up?

2. What instruction was the computer executing when the routine hung up?

3. What was (rA) immediately before the execution of this instruction?

4. What is the word that was being added to (rA) when the routine hung up?

When the computer is on continuous, the only part of the central computer group that moves slowly enough for the mind of the programmer to keep up with is the tape on the Uniservos. This tape movement can usually be predicted from the nature of the routine, and before the debugging run the programmer should figure out and fix in his mind every detail of the expected tape movement. During the debugging run the programmer's main interest should be directed toward the movement of the tapes,not at the SCP. Any deviation from the expected tape movement is usually a good indication of a bug.

THE EMPTY OPERATION

It sometimes happens that, after a bug has been detected, the programmer could profitably utilize a record of the contents of a certain portion of the memory. If the portion is not too large, this record can be printed on SCP by means of the empty operation. The empty operation is initiated by depressing the type out selector button labelled "empty" and operates as follows. The contents of the cell specified by the four least significant digits of (CC) are printed. (CC) are automatically increased by one, and the contents of the next specified cell is printed. The process can be continued until the contents of all cells wanted by the programmer are printed.

MEMORY DUMP

If the portion of the memory, a record of which the programmer wants, is too large to be printed in a short amount of time, a memory dump can be used to obtain the record. Memory dump consists of writing the contents of the memory on tape in order that the tape can be printed on the High-Speed Printer. To produce the memory dump, a routine that will write the contents of the memory on a tape is coded, unityped on the instruction tape, and read into the memory at the same time as the routine to be debugged is read. When a memory dump is desired, control is transferred to the memory dump routine by means of an SCICR. It is standard debugging procedure to obtain a memory dump whenever a bug occurs and cannot immediately be corrected.

VERIFYING THE OUTPUT

If a routine runs through the debugging run to completion, and the programmer has been unable to detect any bugs from the tape movement, the output of the routine must then be checked to verify that it is the output expected from the given input. The verification can be done visually by printing the output on the High-Speed Printer. However, it is often possible, especially if the input data has been generated, to code a routine that will accept the output of the routine to be debugged as input, and compare it with the expected results. Such a checking routine usually prints all pertinent information about any discrepancies on the SCP.

SUMMARY OF PROCEDURES TO FOLLOW FOR TEST RUNNING A ROUTINE

A. Prior to running the routine

1. Prepare a detailed memory allocation including working storage.
2. Prepare detailed operating instructions including:

   a. servo allocation - inputs, instructions, blanks
   b. a description of SCP printouts and necessary type-ins
   c. breakpoints included in routine - how and when used
   d. a list of servo buttons to be depressed
   e. the disposition of output

3. Code a data generator and a checking routine if applicable
4. Thoroughly desk check the routine
5. Determine the nature of tape movement

B. To run the routine

1. Mount tapes
2. Inform the computer operator of buttons and switches to be used

3. Initial Read the instruction tape
4. Place computer on continuous

C. While the routine runs

1. Observe tapes for characteristic movement
2. Listen for characteristic sound of a closed loop or stoppage.

D. If the computer stalls

1. Determine the type of error (neons lit, SR, CR)
2. Examine the contents of affected registers and memory cells (type-outs, empty, etc.)
3. Determine the location of the error (type out (CC))
4. If the error can be corrected and the routine continued, do so. (type-ins, fill, etc.)
5. If necessary, write the memory on tape (the coding to do this should be in the routine - or fill the coding)
6. When appropriate,employ service routines to locate the source of the error.
7. Desk check the routine and list the corrections to be made.

E. If there is a closed loop in the routine

1. Depress "all" breakpoint selector button
2. Depress start bar (as many times as is necessary) noting the Qm and Tm instructions and the condition of the conditional transfer neons.
3. When a pattern is determined proceed to D3, above.
4. If there are no Qm's or Tm's in the loop,execute the loop one instruction at a time.

F. When tape movement is not as expected

1. Stop computer
2. Proceed to D5, above.

G. When the routine runs completely, check the output.

SUMMARY

Interrupted Operation: by depressing appropriate buttons, instructions, either from the program or set up in SR by means of switches, can be executed.individually.

SCICR: type one word into CR on SCK.

Fill: in a continuous operation, any number of words may be typed into consecutive memory locations.

Empty: in a continuous operation, any number of words, in consecutive memory cells, may be printed on SCP.

Retain Instruction: a particular stage of the 3 stage cycle may be retained to perform manual operations. This prevents "losing one's place" in a routine.

Retain C: alteration of (CC) is prevented. This has a function similar to Retain Instruction.

50m: may print (rA), (rL), (rX), (rF), (CC), or (CR) depending on the selection of the Output Selector Button.

BREAKPOINTS

Qnm, Tnm: stops computer, in conjunction with Conditional Transfer Selector Buttons, 0-9 and All, after the comparison had been made. The result of the comparison is shown in two neons and may be altered by two buttons.

,0m: stops computer where Comma Breakpoint Switch is depressed; interpreted as a skip otherwise.

50m: will stop the computer, be treated as a 00m, or as a normal 50m depending on the position of the Typeout Breakpoint Switch.

chapter 12

Sorting and Merging

A characteristic of most commercial data processing is the necessity of assembling with a master file one or more files containing additional information about the items before their processing can begin. Each item of the master file as well as the subsidiary files are usually identified by a key contained within the item. For example, payroll items are identified by a badge number; inventory records, by a stock number. The assembly process consists of selecting an item from the master file and then selecting from the subsidiary files items with a matching key.

The central problem in such matching operations is the cost at which the information for a given item can be selected from the subsidiary files. The speed at which this selection can be performed will affect this cost. The longer the selection takes, the more equipment and operative personnel are necessary to accomplish it within the time available.

In investigating the speed of selecting an item, define access time as that time necessary to select a given item when that item is one of many items in a file. The given item may be chosen in two ways, with each manner of choice giving a different access time. Random access time is the time required to select an item designated in random fashion. Sequential access time is the time required to select item k+1 after having already selected item k. The random and sequential access times for the memory are essentially the same.

The files of most commercial applications are of very large size. The master file for a 10,000 man payroll might contain 12,000,000 bits of information, and this is a relatively small file compared to the million accounts in the master file of a public utility company. Yet only a part of such a file could be stored in the memory. The cost of memory storage prohibits the expansion of the memory.

The cost of storing files on tape is economically feasible. The random access time of tape files depends on the number of items stored, but even for 27,000 ten word items this time needed to select one item is over one and one half minutes. The sequential access time for such a file is only about ten ms.

The initial order of one or more files in an application is usually not controllable. For example, the receipt of payments never come in account number sequence.

With tape storage items should be arranged in a sequence to take advantage of the short sequential access time. The usual sequence is an ascending one. Sorting is that process which arranges a tape of items in sequence by a given key.

COLLATION

Collation combines two or more similarly ordered sets of items to produce another ordered set composed of information from the original sets.

Assume a random sequence of items on tape. Each item has a four character alphanumeric key by which the items are to be arranged in ascending sequence. In the following figure the tape is shown on Univac Uniservo 2. Only the key of each item is shown. Blank tapes are mounted on Uniservos 3, 4, 5 and 6.

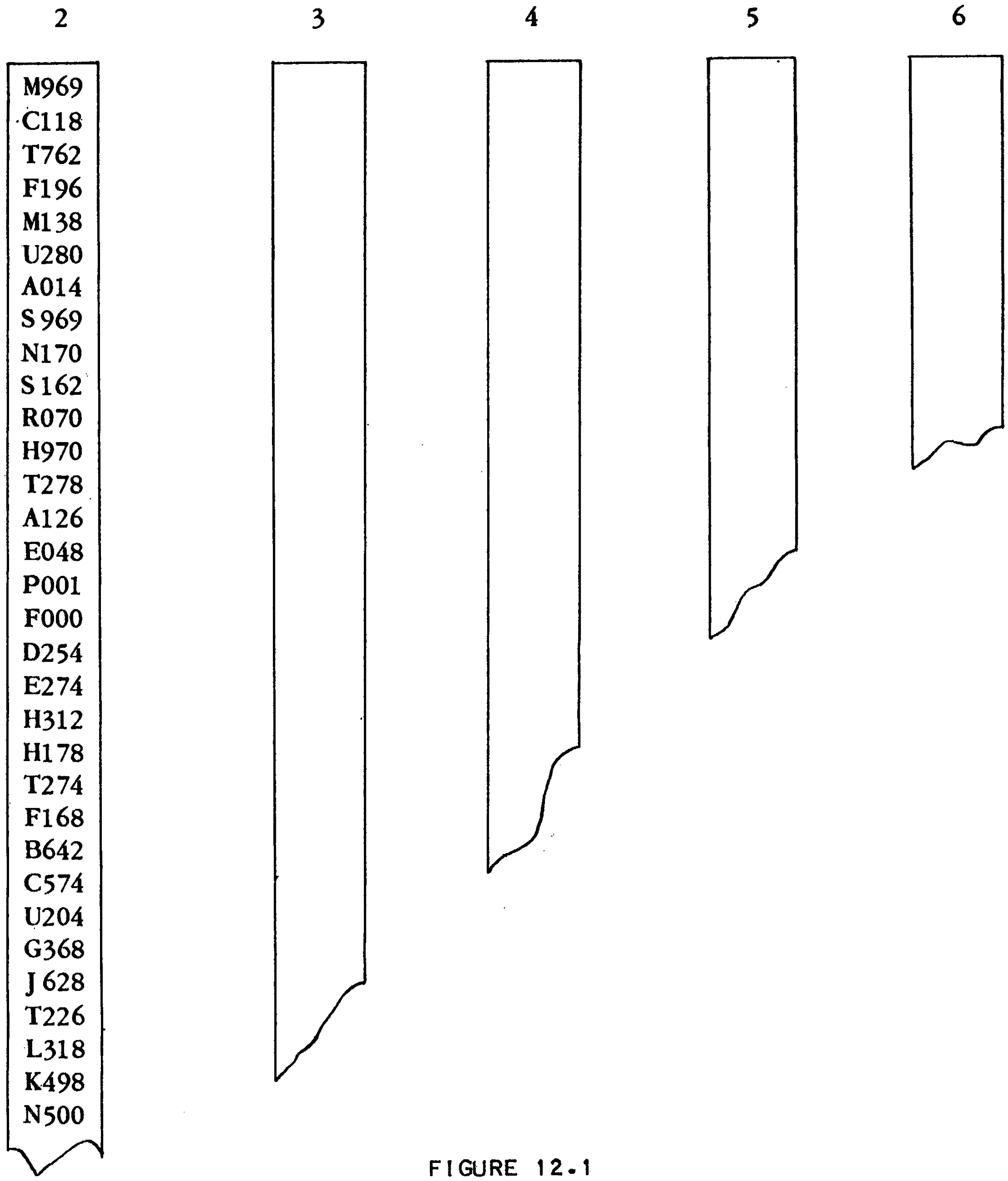
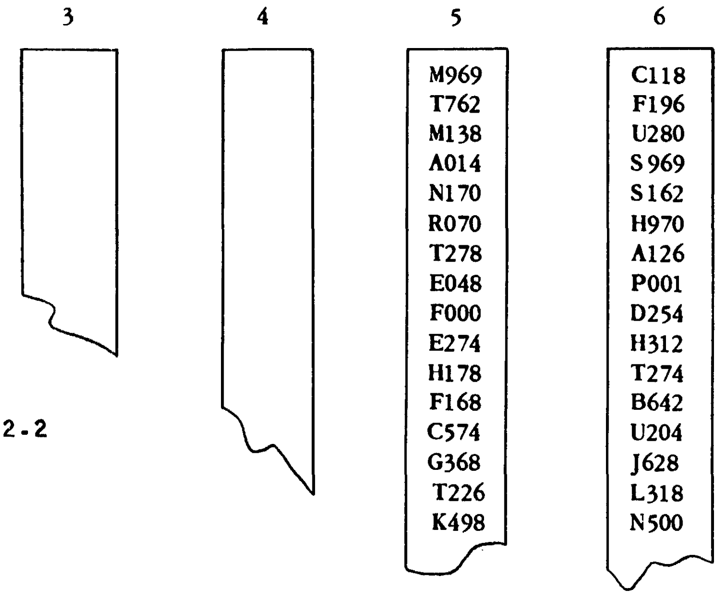


FIGURE 12.1

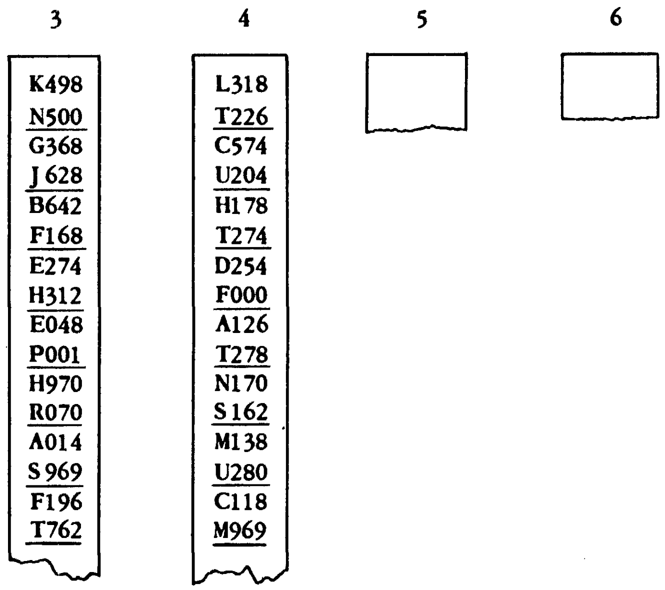
The first step is to read down tape 2, and write the items alternately on tapes 5 and 6.



3	4	5	6
		M969	C118
		T762	F196
		M138	U280
		A014	S969
		N170	S162
		R070	H970
		T278	A126
		E048	P001
		F000	D254
		E274	H312
		H178	T274
		F168	B642
		C574	U204
		G368	J628
		T226	L318
		K498	N500

FIGURE 12-2

Tapes 5 and 6 are now said to consist of one item strings. Next a merge is performed on the strings. Tapes 5 and 6 are read backwards. The first items from each tape are written in ascending order on tape 3. The second items are written in ascending order on tape 4. The process is continued, paired items being written alternately on the output tapes. At the end of the merge, tapes 3 and 4 contain two item strings, as shown in the following figure, where lines are used to indicate the separation of the strings.

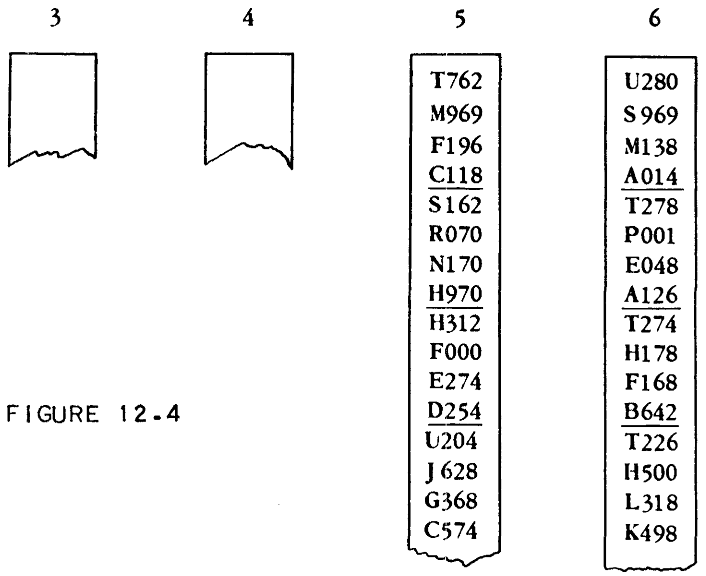


3	4	5	6
K498	L318		
N500	T226		
G368	C574		
J628	U204		
B642	H178		
F168	T274		
E274	D254		
H312	F000		
E048	A126		
P001	T278		
H970	N170		
R070	S162		
A014	M138		
S969	U280		
F196	C118		
T762	M969		

FIGURE 12-3

Tapes 5 and 6 are shown as blank to indicate that they are rewound and may be used to store other data.

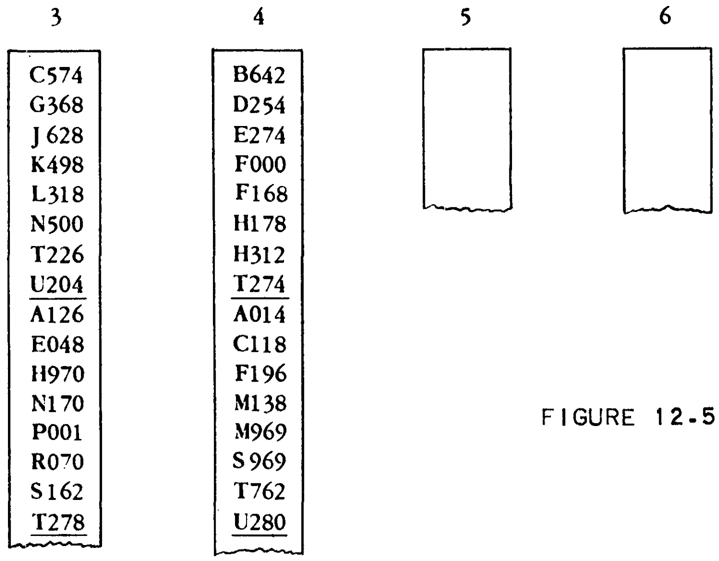
Tapes 3 and 4 are read backwards, the first strings on each tape being merged into a four item string in descending order which is written on tape 5. The second strings are merged and written on tape 4, and so forth.



3	4	5	6
		T762	U280
		M969	S969
		F196	M138
		C118	A014
		S162	T278
		R070	P001
		N170	E048
		H970	A126
		H312	T274
		F000	H178
		E274	F168
		D254	B642
		U204	T226
		J628	H500
		G368	L318
		C574	K498

FIGURE 12-4

The four item strings are merged into eight item strings.



3	4	5	6
C574	B642		
G368	D254		
J628	E274		
K498	F000		
L318	F168		
N500	H178		
T226	H312		
U204	T274		
A126	A014		
E048	C118		
H970	F196		
N170	M138		
P001	M969		
R070	S969		
S162	T762		
T278	U280		

FIGURE 12-5

The eight item strings are merged into 16 item strings. Since there are 32 items, there are two strings.

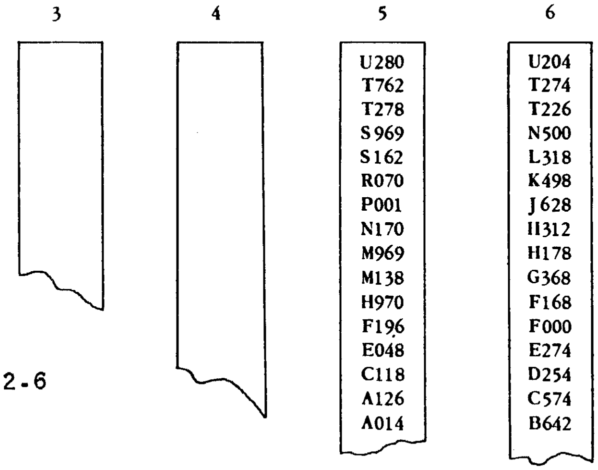


FIGURE 12-6

The last pass merges the 16 item strings onto tape 3.

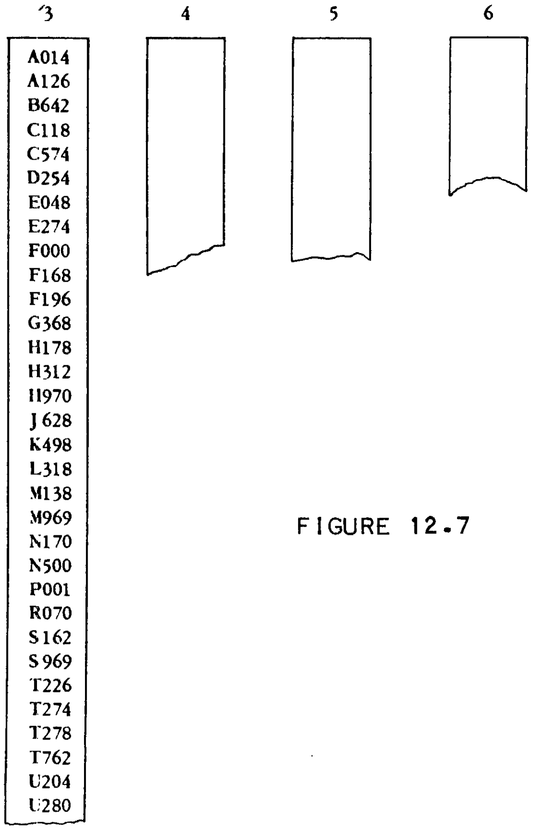


FIGURE 12.7

Thus, collation consists of successive string merges. Each merge doubles the string length and halves the number of strings. The same number of passes sort the items regardless of the size key involved.

When collation is done on the computer, each block of input to the first pass is sorted in the memory, and one block strings are dispersed. Thus, if two word items were to be sorted, the first pass would disperse 30 item strings.

To guarantee that the last pass will produce an ascending sequence, the following rule is used. Given k blocks of items, determine n such that

$$2^{n-1} < k \leq 2^n, \quad k \neq 1$$

If n is odd, the first merge should be ascending; if even, descending.

The above is called two way collation. The minimum number of Uniservos required is four, since once the instructions have been read and the data dispersed, the instruction and input data tapes can be dismounted, freeing their Uniservos for blanks. The number of passes to sort a tape containing 4500 blocks is 14. When enough Uniservos are available, which is usually the case, three way collation is more efficient. Three way collation proceeds exactly as two way except that the first pass disperses over three tapes, thus permitting a series of three way merges. Consequently, the number of blocks in the strings increase by powers of three rather than two as in two way collation. In three way collation the sequence of the first merge is determined by the same method used in two way except that the formula becomes

$$3^{n-1} < k \leq 3^n, \quad k \neq 1$$

Using three way collation, the number of passes to sort a tape containing 4500 blocks of items is nine.

DIGITAL SORT

While collation provides a powerful general sorting method, there are occasions when the nature of the items to be sorted permits specialized techniques that, for those cases only, are more efficient. One such technique is the digital sort. A digital sort is feasible when the key is numeric and small.

Assume a random sequence of items on Uniservo 2. Blank tapes are mounted on Uniservos 3, 4, 5, 6, 7, 8, 9, -, A, B and C.

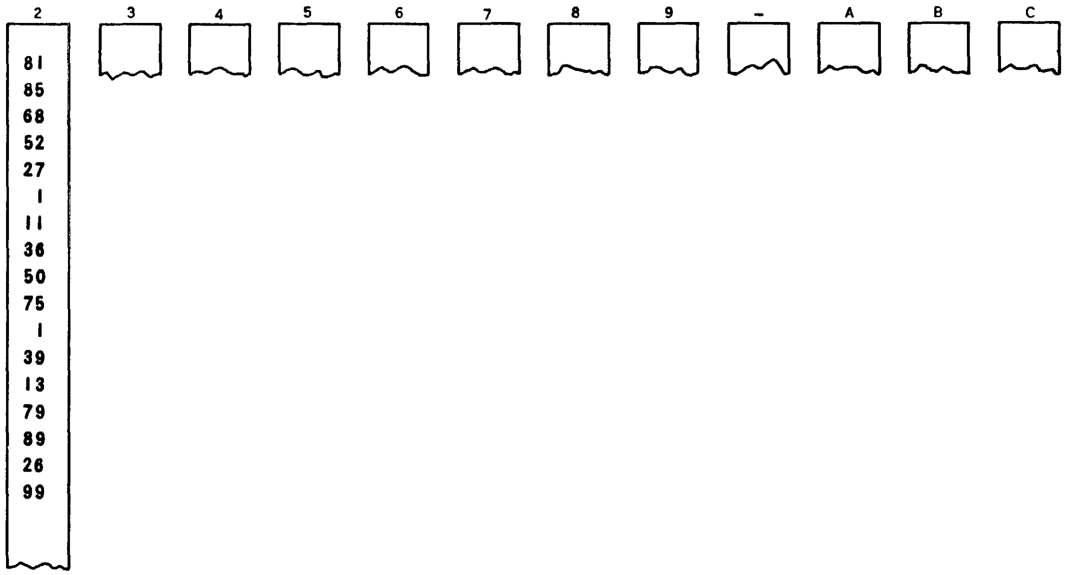


FIGURE 12-8

The dispersion run reads down the data and examines each key. If the least significant digit of the key is zero, the item is written on tape 4; if one, on tape 5,; if two, on 6; and so on.

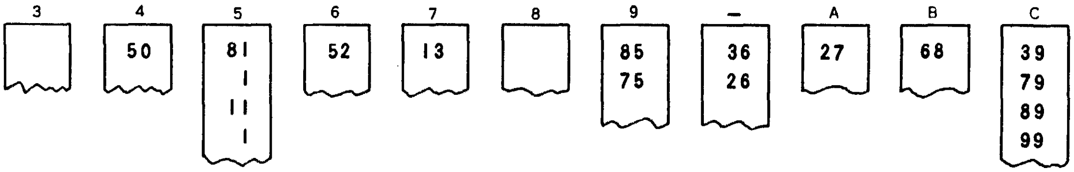


FIGURE 12-9

The next step is the collection. Tape 4 is read backwards. Each 0 item is recorded on tape 3. After the 0 items have been recorded on tape 3, the 1 items are selected from tape 5, again reading backwards, and written on tape 3. This process is continued until all the items have been collected on tape 3. The items are now arranged in ascending order by the least significant digit of the key.

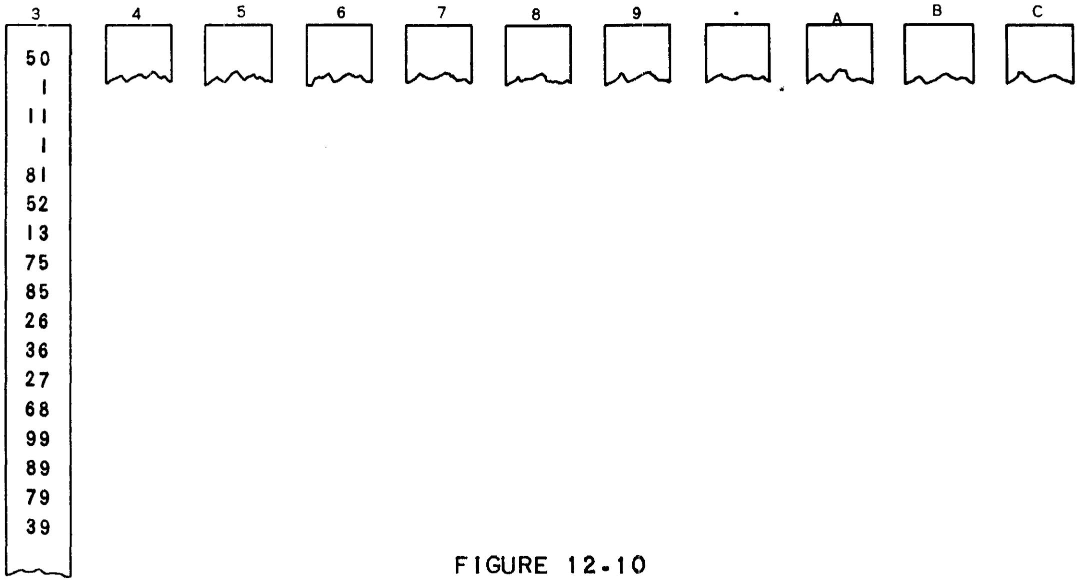


FIGURE 12-10

Tape 3 is now read backwards to repeat the dispersion run, but this time the digit in the second column of the key is examined.

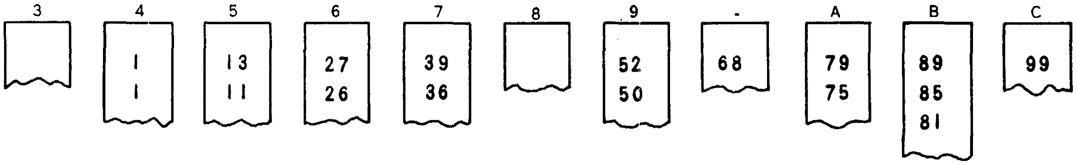


FIGURE 12-11

Next the collection run is repeated. The items are now sorted.

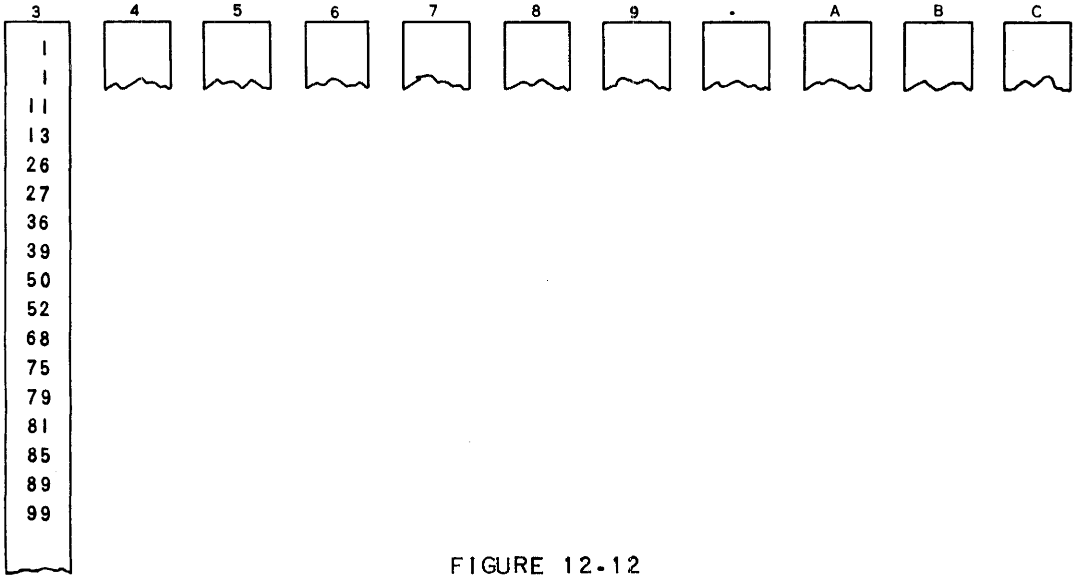


FIGURE 12-12

The digital sort proceeds in this fashion for each additional digit in the key. Using the digital sort, the numbers of passes to sort a tape is twice the number of digits in the key. The time required for each pass is about the same as for the collation method on like size items, since the same general operations are needed. The number of passes required in this example is four as compared with nine for three way collation. Of course, the digital sort is not of universal application, but on sufficiently restricted data it is efficient. Thus, the nature of the items to be sorted should be given careful study to determine if a special sorting technique is applicable before recourse to the general method of collation is made.

FUNCTION TABLE SORT

Another specialized sorting technique is the function table sort. This technique is feasible when the following conditions are met.

1. The key is numeric.
2. The range of the key is small.
3. All items having the same key can be summarized in a single item.
4. The item size is small.

Assume a series of two word items with keys ranging from 0 to 4999.

The first run reads the items on tape 2 and disperses them on tapes 4-8. Those items with keys 0-999 are written on tape 4; items with keys 1000-1999, on tape 5; and so on.

The second dispersion run reads tape 4 backwards and distributes the items between tapes 5 and 6, the items being written following the first dispersion items. Sentinel blocks may be used to mark the separation between the two sets of items. Items with keys 0-499 are written on tape 5; the others, on tape 6.

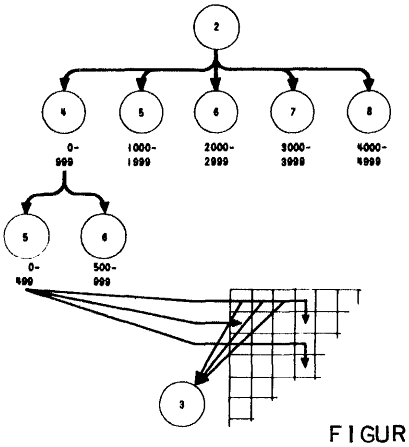


FIGURE 12-13

The third step is the function table sort proper. One thousand consecutive cells are reserved in the memory. These cells will store 500 items; they are cleared to zero initially. The items with keys 0-499 from tape 5 are read into the computer. The key of each item determines the cells in which the item is to be placed. Cells 1000 and 1001 are reserved for items with key 0, 1002 and 1003 for items with key 1, and so forth. The address, m, is determined from the formula

$$m = 1000 + 2\ (k - 250n)$$

where k is the key and n = 0 initially. Then the item is transferred with a V2mW2m instruction pair to cell m and m + 1. If more than one item with a given key is possible, the quantity fields in the items are added to the quantity fields already stored in the paired cells.

Thus, as each item is read into the computer an address is fabricated from its key and the item stored in that cell and the one succeeding. If an item is already in the cell the quantity field is added to the item stored. The process is akin to a post office pigeonhole set-up where letters selected at random are stuffed into the proper route box. After all the 0-499 items have been stored in their appropriate cells the computer "looks" in cells 1000 and 1001. If an item is present, it is written on the output tape and cells 1002 and 1003 are examined. If no item is stored here, cells 1004 and 1005 are examined. In this manner the items which were stuffed into the proper pigeonholes in random fashion are now extracted and written on tape 3 in sequence.

After all 500 paired cells have been examined, they are cleared to zero, and the items from tape 6 with keys 500-999 are read into the computer. These items are stored in the appropriate cells, the address being fabricated from the formula given above, with n = 1. Thus, the item with key 256 is stored in cells 1012 and 1013. Again, after all these items have been stored, the cells are emptied sequentially and their contents written on Uniservo 3. Then the second dispersion and function table runs are repeated for each of the items on tapes 5, 6, 7 and 8. Tape 3 now contains the sorted items.

Only four passes over the information are required: one for first dispersion, one for second dispersion and two for the function table sort (one pass to store the items, one pass to pick them out of the cells and write them on tape 3). To do this sort by three way collation would require nine tape passes; by the digital sort, eight tape passes. The number of passes needed by a function table sort can be determined by the following formula.

$$P = 2 + \left(\frac{wk}{cu} + .999...\right)_{IP}$$

Where p - number of passes
w - number of words in the item
k - range of the key
c - number of cells available for the function table sort
u - number of Uniservos available for dispersion

The sorting methods described sort one reel of data only. Where the amount of data to be sorted exceeds one tape, a merge must be done to produce the multi-reel ordered items. For example, if three reels of data are to be sorted, the three reels are first individually sorted by that sorting method most efficient for this data. Then the three sorted reels are put through a three-way merge to form a single ordered file of three reels.

The merging time is not trivial for large amounts of data. Generally speaking, the merging time depends on the number of Uniservos available and the number of full reels to be merged. The objective in sorting and merging is to keep the number of passes to a minimum. Because of this it is desirable to have the maximum number of input reels to merging runs as the Uniservos available will allow.

Care should be taken in the manner in which merging operations are performed. As an example, consider the merging of 12 reels with the number of Uniservos permitting a maximum of 3-way merging. The straightforward approach would be to do four 3-way merges producing four ordered piles of three reels each. Then three of these piles would be merged to form one ordered pile of nine reels which is then merged with the fourth three-reel pile to form one pile of twelve reels. The number of tape passes is 33.

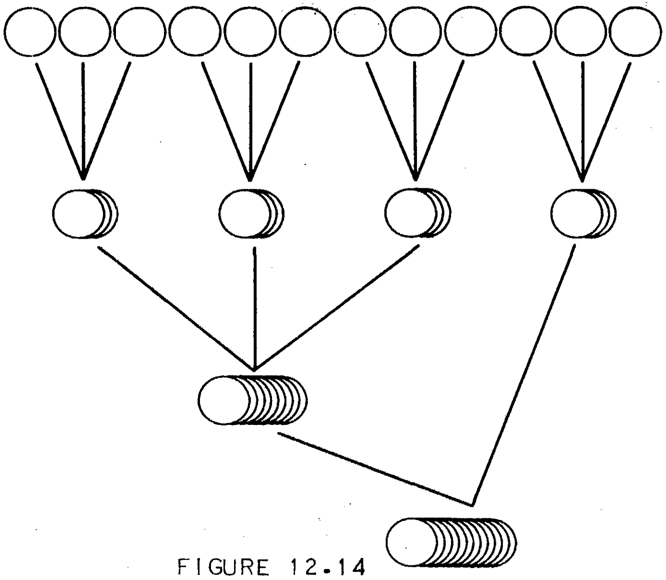


FIGURE 12-14

However, by merging in the manner shown in figure 12-15, only 29 passes are involved.

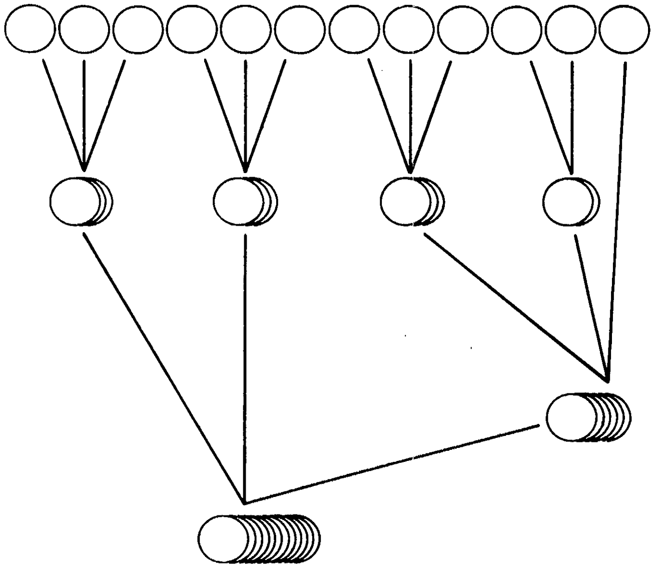


FIGURE 12-15

An algorithm exists for expressing the exact manner in which the merging may be done for minimum tape passes. The algorithm is as follows. Given k tapes to merge and enough Uniservos to do a b way merge find n such that

$$b^{n-1} < k \leq b^n$$

Form b groups, G_1, G_2, G_3,...G_b, each consisting of b^{n-2} tapes. Add the remaining tapes to the groups according to the following rules.

1. No group may contain more than b^{n-1} tapes.
2. Bring G_1 to its maximum first, then G_2; then G_3; ...; and finally G_b.

To use the algorithm:

A. Decide on b- commonly 3, 4, or 5.
B. Establish n using k and b.
C. Word backwards from k, fixing the size of the groups following rules 1 and 2, above.

Applying the algorithm to the 12 reels:

$$\left.\begin{array}{l} k = 12 \\ b = 3 \end{array}\right\} \quad 3^{n-1} < 12 \leq 3^n$$

$n = 3$

$b^{n-2} = 3$

The groups are:

INITIALLY	FINALLY	
G_1 :3	6	(Maximum G= 9)
G_2 :3	3	
G_3 :3	3	

These, in turn, must be taken as new k's. G_2 and G_3 result from 3 way merges of single reels. G_1 is derived using the algorithm with $k = 6$ to give

$$\left.\begin{matrix} k = 6 \\ b = 3 \end{matrix}\right\} 3^{n-1} < 6 \leq 3^n$$

$$n = 2$$

$$b^{n-2} = 1$$

INITIALLY	FINALLY	
G_1 :1	3	(Maximum G=3)
G_2 :1	2	
G_3 :1	1	

Thus, the six reels are formed by a 3 way merge and a two way merge of single reels, and then a three way merge of the three reel string, the two reel string and the single reel.

chapter 13

Preparation and Disposition of Data

INPUT UNITS

The Central Computer of the Univac Data-Automation System efficiently accepts large volume data only from tape; therefore, all such data is prerecorded on this medium. In addition to computer recording, three other means are available for recording tape.

1. Keyboard to tape recording.
2. Card-to-tape recording.
3. Paper to magnetic tape recording.

KEYBOARD TO TAPE RECORDING

UNIVAC UNITYPER

The Univac Unityper is keyboard operated and records each key stroke on tape while also producing a printed copy. It is the primary device for recording source documents on tape. The Unityper is desk size and consists of a modified electric typewriter containing a recording head, a tape transport mechanism and housing unit, and a power supply. The keyboard is similar to the standard typewriter keyboard with the following modifications.

1. In addition to the standard numeric keys, there is a special set of 10 numeric keys arranged to facilitate more rapid recording of numerical data.
2. All alphabetics are printed as capitals.
3. Special keys are available for representing characters peculiar to the Univac Computer code and for controlling the operation of the Unityper.

The Unityper prints 120 characters to a line, each printed line being recorded on tape as a blockette at a density of 50 characters per inch. A blockette is a group of ten words. A space of 2.4 inches is left between blockettes. Any errors made while typing a blockette, as evidenced in the printed copy, can be corrected: singly, by backspacing the tape to the error and retyping the blockette from that point; or for a complete blockette, by depressing the Erase Key, causing the whole blockette to be erased and the tape to be positioned for retyping.

In some cases, the data to be recorded may not completely fill a blockette, or it may be desirable to simplify the computer processing by insertion of spaces or zeros between fields. Special Unityper keys provide for automatically filling a blockette, or portions of a blockette, with zeros or spaces. This is done by first setting the Fill Selector Switch to either the space or zero position. Then when the Fill Key is depressed the carriage will be advanced either to the next tab stop or to the end of the line, if no tab stops have been set. The character chosen by the Fill Selector Switch is recorded on tape in the positions transversed by the carriage. The average recording rate on the Unityper is 10,000 characters per hour.

UNIVAC VERIFIER

The main function of the Univac Verifier is to verify the correctness of tapes prepared on the Unityper. In addition, the Verifier can be used to prepare tapes in the same way in which the Unityper is used.

The Verifier consists of three units housed in a standard size typist's desk. The units are the typewriter unit, the tape reader unit, and the control and checking unit.

Verification consists of comparing, digit by digit, the data on a Unityped tape with a second typing of the source document. A printed copy, produced on the typewriter unit, records the actions performed in the verification process.

The Verifier's tape reader reads, and sets up in the thyratron memory of the control unit, the first character on tape. The operator then strikes the key of the first character on the source document. If the character of the key struck and the character on tape agree, the typewriter prints the character in red. If there is a disagreement between the characters, the character is printed and then the keyboard locks. The determination of what the error is can be made by backspacing and viewing the character from tape on a neon display. The character on tape can be changed by use of the Correct Key, or if correct, may be reverified to continue the operation. If an entire blockette requires correction, the Change One Line Key is used. Both of these keys will switch the Verifier's function temporarily to recording.

As each character is transferred from tape to the Verifier's memory it is counted. More or less than 120 digits from a blockette will stop the Verifier with the digit count error neon lit.

The maximum rate of verification is 12 characters per second. Nonsignificant information can be skipped without printing or verifying at the rate of 80 characters per second.

PUNCHED CARD-TO-MAGNETIC TAPE RECORDING

UNIVAC 80-COLUMN PUNCHED CARD-TO-MAGNETIC TAPE CONVERTER

The Univac 80-Column Punched Card-To-Magnetic Tape Converter is a device for automatically recording data from 80-column punched cards on tape. The card to tape conversion is a checked operation. The rate of conversion is 240 cards per minute. Each card is recorded as a blockette. The Converter consists of three cabinets, the tape cabinet, the card reader cabinet and the control and memory cabinet.

A card is initially read at the first reading station of the card reader, and the data is stored in the magnetic core memory of the control cabinet. As the data is read it is edited by a plugboard. The edited data is then written on tape.

The tape is then read back to the beginning of the blockette just written. As this is being done, a second reading is made of the card. Each column is read at a different reading station from that of the first reading and stored in a different position in the memory. The blockette is then read forward, and a comparison is made between the tape recording and the second card reading in the memory. During this comparison, and as the tape is read back, each character is counted and its binary code checked. If an even binary code or a digit count error is present, or if there is disagreement between the tape and card recordings, the card will be ejected into an error bin, and the tape will be repositioned at the beginning of the faulty blockette for rerecording. When this occurs, the operator has the following choices of action.

If the sequence of cards must be maintained on tape, the error card may be reinserted in the card reader at the head of the cards and the conversion continued. If the error was transient, the card should be converted successfully, but if the card again fails to convert, an adjustment may be necessary.

If card sequence is not important, the error cards can be accumulated till the end of the run, reinserted in the card reader, and converted in a group.

If all checks pass, the card counter will be stepped and the next card converted. The failure to feed a card is automatically detected by requiring each card fed to generate the signal which causes the next card to be fed.

The 80 characters of each card may be rearranged in any way in the 120 character blockette by the wiring of a detachable plugboard. If desired, up to 24 overpunched columns (X or Y) on a card may be separately recorded as a minus and ampersand, respectively, for the overpunches, and as the corresponding numeral for any other punch in the column. The overpunch symbols may be distributed anywhere in the blockette. Thus, the data may be spread over as many as 104 characters within each blockette. Unused characters of the blockette and unpunched columns in the card are recorded as zeros or space symbols as determined by the setting of the Blank Column Selector, a special plugboard control. The method of complement plugging is used as a check on the correct functioning of the plugboard during conversion. This method requires all wires of the plugboard to emit a continuous signal throughout the conversion.

The 80-Column Card-To-Magnetic Tape Converter can accept combinations of punches representing 26 alphabetics, 10 numerals and 12 miscellaneous symbols.

All the acceptable card punches and their corresponding Univac Computer characters are listed below.

CARD PUNCH	UNIVAC COMPUTER CHARACTER	CARD PUNCH	UNIVAC COMPUTER CHARACTER
No Punch	Δ or 0 (Determined by	12-8	H
12	& blank column	12-9	I
11	- selector)	11-1	J
0	0	11-2	K
1	1	11-3	L
2	2	11-4	M
3	3	11-5	N
4	4	11-6	O
5	5	11-7	P
6	6	11-8	Q
7	7	11-9	R
8	8	0-1	/
9	9	0-2	S
12-1	A	0-3	T
12-2	B	0-4	U
12-3	C	0-5	V
12-4	D	0-6	W
12-5	E	0-7	X
12-6	F	0-8	Y
12-7	G	0-9	Z

Some punched card installations make use of triple punched columns, known as the 407 code. A slight modification of the 80 Column Converter, an optional feature, will translate these triple punches into Univac Computer characters, as shown below.

CARD PUNCH	UNIVAC COMPUTER CHARACTER
3-8	#
4-8	@
Y-3-8	.
Y-4-8	:
X-3-8	$
X-4-8	*
O-3-8	,
O-4-8	%

Unless the triple punch modifications are present, the 80 Column Converter will interpret triple punched card columns as mispunches, and will eject the triple punched card into an error bin.

UNIVAC 90-COLUMN PUNCHED CARD-TO-MAGNETIC TAPE CONVERTER

The Univac 90-Column Punched Card-To-Magnetic Tape Converter is a device for reading data from 90-column punched cards and recording it on tape. The differences between the 90 and 80-Column Converters are as follows. In all other respects the Converters are identical. The card data may be spread over as many as 114 characters of the blockette. The 90-Column Card-To-Magnetic Tape Converter can accept the combination of holes representing 26 alphabetic symbols, 10 numerals and 7 miscellaneous symbols. All of the acceptable card punches and their corresponding Univac Computer characters are listed below.

CARD PUNCH	UNIVAC COMPUTER CHARACTER	CARD PUNCH	UNIVAC COMPUTER CHARACTER
no punch	Δ or 0 (Determined by	3-7	H
0	0 blank column	3-5	I
1	1 selector)	1-3-5	J
1-9	2	3-5-9	K
3	3	0-9	L
3-9	4	0-5	M
5	5	0-5-9	N
5-9	6	1-3	O
7	7	1-3-7	P
7-9	8	3-5-7	Q
9	9	1-7	R
1-5-9	A	1-5-7	S
1-5	B	3-7-9	T
0-7	C	0-5-7	U
0-3-5	D	0-3-9	V
0-3	E	0-3-7	W
1-7-9	F	0-7-9	X
5-7	G	1-3-9	Y
		5-7-9	Z

If cards containing 4 or more punches in any column are fed into the 90-Column Converter, they will be ejected into an error bin, unless the modified Converter is used. The modified Converter permits cards to be converted which contain 4 or more punches as follows.

CARD PUNCH	UNIVAC COMPUTER CHARACTER
1-3-5-7	)
1-3-5-9	.
1-3-7-9	:
1-5-7-9	+
3-5-7-9	/
1-3-5-7-9	;

PAPER TO MAGNETIC TAPE RECORDING

The Univac Paper-To-Magnetic Tape Converter, PTM, translates the five, six, or seven level code of perforated paper tape to magnetic tape. The PTM consists of three components housed in a single cabinet: the paper tape reader, the translator and the control unit.

The paper tape reader reads the paper tape code into the translator unit at the rate of 200 characters per second. As each character enters the translator it is converted into the Univac Computer code. The translated characters are then stored in a 120 character memory. When the memory is filled the 120 characters are recorded on tape as a blockette at the density of 128 characters per inch; a space of an inch is left between blockettes.

OUTPUT UNITS

The computer efficiently produces large volume data only on tape. Three means are available for converting data on tape to some other form of output.

1. Tape to printed copy.
2. Tape to punched cards.
3. Magnetic to paper tape.

TAPE TO PRINTED COPY

UNIVAC HIGH-SPEED PRINTER

The Univac High-Speed Printer is a device for large volume printing of data. The standard printing speed is 600 printed lines per minute, with up to seven legible carbons. The Printer accepts paper from 4" to 27" in width and up to card stock in thickness, and has a 130 character printing line. Paper may be preprinted and serrated. There are 51 printable characters: 26 alphabetics, 10 numerics and 15 miscellaneous symbols: # $ % * () / - + : ; . , ' and &.

Tapes recorded in blockette form at densities up to 128 character per inch with a minimum of one inch between blockettes are acceptable to the Printer. These tapes include tapes produced by the Unityper, the Verifier, the Card-to-Tape Converters, the PTM, and the computer. The computer writes a tape for the High-Speed Printer as follows.

On the Supervisory Control Panel are a series of 16 buttons, called Block Subdivision Buttons and labelled with the names of the Uniservos. If a Block Subdivision Button is depressed, all writing done on the corresponding Uniservo will be in blockette form. The space between blockettes on Uniservos 8, 9 and - will be one tenth of an inch, on all other servos, one inch.

Also on the Supervisory Control Panel are a similar series of buttons called Tape Density Buttons. If a Tape Density Button is depressed, all 5nm instructions executed with respect to the corresponding Uniservo will write, not at a density of 250 characters per inch, but at 124 per inch.

The High-Speed Printer is housed in four cabinets, the tape cabinet, the printing cabinet, the control and checking cabinet, and the power supply cabinet.

Through the use of a detachable plugboard, the horizontal format for each blockette printed can be set up in such a manner that

1. any character of the blockette can be printed in any one of the 130 print positions,
2. fields of the blockette can be printed on as many as six consecutive lines.

and 3. fields of the blockette can be printed as many as three times on any or all of the six consecutive lines.

The plugboard also enables the suppression of the printing of nonsignificant zeros in a numeric field.

The vertical format of printing is regulated by a 7 channel punched paper loop located in the printing cabinet, which advances in synchronism with the paper. The sensing of holes in certain channels of this loop will cause the paper feed to either fast feed the paper or else to discontinue a fast feed presently in progress. No printing occurs while the paper is being fast fed. There are two ways in which a fast feed can be initiated: by a symbol on tape or by a hole in the paper loop.

As a blockette is read from tape to the memory, each character is counted. More or less than 120 characters in a blockette stops the Univac High-Speed Printer and lights the character count error neon.

As each character is transferred from tape to the memory, and from the memory to the comparator, it is given an odd even check. An illegitimate character code stops the High-Speed Printer and lights the odd even error neon.

The Univac High-Speed Printer also checks against

1. the failure of a character to print
2. the printing of more than one character in a print position

and 3. the printing of a character other than the character meant to be printed.

The occurrence of any of the above stops the High-Speed Printer and lights an appropriate neon.

MAGNETIC TAPE TO PUNCHED CARDS

The Univac Magnetic Tape-to-Card Converter transfers data from magnetic tape to 80-column punched cards. Input to the Converter is tape recorded in blockette form, a space of 1/10 inch between blockettes. An 80-column card is punched from selected portions of each blockette. The conversion is checked and proceeds at 120 cards per minute. The Converter consists of three cabinets, the tape cabinet, the card punching cabinet and the control cabinet.

A blockette is read from tape and stored in the magnetic drum memory, located in the control cabinet. The format of the blockette on the drum is controlled by a detachable plugboard. This plugboard is used to select the 80 characters of each blockette for punching and the positions on the card where they are to be punched. Any character can be punched in any column.

The edited blockette, in the drum memory, is sent to the card punch to be punched. Columns which are not plugged on the plugboard are not punched. After a blockette has been punched, the next blockette, having been read and edited during the punching of the preceding blockette, is sent to the card punch.

The conversion continues in this manner until a blockette containing a printer stop symbol is read. The blockette containing the printer stop is not punched.

As a blockette is read from tape to the Converter's memory each character is counted. If this count is other than 120, the Converter stops with the character count error neon lit.

As each character is read from tape to the memory its code is checked. If a character with an even number of pulses in its code is present, the Converter stops with the Digit Odd-Even Error Neon lit.

After each card is punched it is read at a second station in the punch unit. This data is stored in a special section of the memory. A character by character comparison is then made between the data punched on the card and the data originally read from the tape. If any inequalities are detected, the card punched is ejected into an error bin, and the Converter stops with the appropriate error neon lit.

As the card data is sent to the card punch each character's code is checked. If a character with an even number of pulses in its code is detected, the Converter stops with the appropriate error neon lit. If an) of the above errors occur, the Converter can be restarted to either reread the blockette or repunch the card. If the error is transient, the conversion will be successful on the second attempt. The conversion table showing the equivalent tape characters and card punch combinations is shown below.

UNIVAC COMPUTER CHARACTER	CARD PUNCH	UNIVAC COMPUTER CHARACTER	CARD PUNCH
none	12	G	12-7
–	11	H	12-8
0	0	I	12-9
1	1	J	11-1
2	2	K	11-2
3	3	L	11-3
4	4	M	11-4
5	5	N	11-5
6	6	O	11-6
7	7	P	11-7
8	8	Q	11-8
9	9	R	11-9
:	12-0	S	0-2
.	11-0	T	0-3
/	0-1	U	0-4
A	12-1	V	0-5
B	12-2	W	0-6
C	12-3	X	0-7
D	12-4	Y	0-8
E	12-5	Z	0-9
F	12-6	none	Blank

MAGNETIC TO PAPER TAPE

The Univac Magnetic-to-Paper Tape Converter, MTP, translates magnetic tape into the five, six, or seven level code of perforated paper tape. The MTP consists of a magnetic tape reader and a paper tape punch.

The punch operates at 60 characters per second. The MTP automatically punches teletypewriter function codes in the paper tape.

SUMMARY

INPUT UNITS

Univac Unityper: a key driven device which records directly on magnetic tape while producing a typewritten copy of the information.

Univac Verifier: may be used as a Unityper or as a verification device for Unityped tapes.

Univac Punched Card-to-Magnetic Tape Converter: records on tape the information stored in punched cards. One model converts 80-column; and another, 90-column cards.

Univac Paper-to-Magnetic Tape Converter: converts information in punched paper tape to magnetic tape.

OUTPUT UNITS

Univac High-Speed Printer: prints the information recorded on magnetic tape.

Univac Magnetic Tape-to-Card Converter: punches the information recorded on magnetic tape into 80-column punched cards.

Univac Magnetic-to-Paper Tape Converter: punches in paper tape the information recorded on magnetic tape.

chapter 14

System Design

This chapter is concerned with the use of the Univac Data-Automation System in commercial applications, and particularly, deals with the development of process charts.

Some factors to be taken into consideration in the use of the Univac System for data processing are

1. choosing the application,
2. preparing the program,
3. planning the cutover

and 4. future planning.

CHOOSING THE APPLICATION

Generally, the use of computers by business to date hasbeen to apply the computer to applications already being performed and which will guarantee savings. There has been some reluctance among management to underwrite the acquisition of a million to a million and a half dollars worth of equipment and to undergo considerable expense in re-vamping procedures and organization structure to tackle problems which give only a theoretical chance of achieving savings. Many computer evaluation committees state that it is a problem to convince management to utilize computers let alone sell them on changing systems. Companies will tackle payroll, premium billing, and customer accounting applications as justification of the computer, but are reluctant to justify a computer on the chance that a more scientific control of inventory may be able to reduce costs. This is not to say that the solving of business problems through the use of operations research or mathematical techniques cannot give even greater savings than standard accounting methods. It is simply that little experience has yet been had to prove that such savings, phenomenal as they may appear on an Operations Research Report, can actually be achieved. Therefore, selection of the initial applications will usually be made from data processing operations now being done. The following factors should also be taken into consideration in the selection of the initial applications.

The initial applications should be limited to as few as possible which will guarantee reaching the break-even point at the earliest date. Care should be taken not to attempt too ambitious a conversion at the beginning as the experience of the company in applying the computer will be nil. The area attacked should be kept as small as possible so as to concentrate both the available personnel and the inevitable mistakes. This purpose can usually be achieved by selecting as the initial applications those areas involving the greatest volume of work and cost and adding to this the next largest application until the break-even point is reached. The remaining applications can be converted on a more leisurely basis. These conversions will, in general, be more efficient because of the now experienced staff.

PREPARING THE PROGRAM

The chronological steps in preparing a program are

1. process charting,
2. flow charting,

and 3. coding.

Flow charting and coding have been explained. Process charting will now be discussed. A process chart is a flow diagram specifying the input, processing, and output of a data processing system. The determination of the input, processing and output required by the system underlies the development of a process chart.

The basic question is "What is wanted as output from the system?" Consider the area of payroll. One of the outputs of the system will be a paycheck for each employee. Data needed on the paycheck is

1. the net pay,
2. the employee's name,
3. the check number,
4. the date of the check

and 5. the employee's badge number.

The check is only one part of the output. Each employee will also receive a stub on which is listed

1. badge number,
2. gross pay,
3. withholding tax,
4. FICA tax,
5. retirement plan contribution,
6. union dues,
7. hospitalization premium,
8. adjustments to gross and net pay,
9. bond deduction

and 10. miscellaneous deductions.

Writing a periodic stub and check is only a part of the output of the system. Other outputs might be

1. reports to federal, state and local governments,
2. reports of labor distribution,
3. lists of employees who are to receive savings bonds,
4. reports of union dues collected,
5. check registers for bank reconciliation,
6. payroll registers for visual reference

and 7. file of relatively permanent data concerning each employee updated each time paychecks are printed.

An analysis of the outputs reveals that certain data must be introduced into the system. The first piece of output was net pay. Net pay is obtained by subtracting all deductions from gross pay. Gross pay must be available in the system. Gross pay is a function of hours worked and hourly rate of pay. These factors are not the result of computation and must be input to the system.

Hours worked may vary considerably from one pay period to the next. Rate of pay changes at infrequent intervals. Hours worked and rate of pay are examples of two general types of input.

A further examination of the output reveals that additional relatively permanent data must enter the system. This data is called master data and includes

1. employee badge number,
2. name,
3. address,
4. social security number,
5. number of dependents,
6. union dues,
7. hospitalization premium,

and 8. bond deduction.

The second type of data is called transaction data. In the case of payroll the transaction data is the clock card data and contains

1. employee badge number,
2. hours absent - medical,
3. hours absent - laid off,
4. hours absent - personal,
5. hours worked,
6. information on shift differentials

and 7. overtime data.

The company contemplating the use of a computer will have already organized its operations to some degree. One of the first tasks, is to obtain "process charts" of the existing system if such charts exist. If not, charts must be drawn.

These charts are used to list the data which is currently the input and output of the system, the frequency of updating which takes place in the files and the format in which the data appears. This information provides the base on which development may proceed. It is not intended, nor is it always desirable, that the computer system be a copy of the existing system.

Many existing systems show some of the following disadvantages.

1. Duplication of information.
2. Unnecessary reports.
3. Unused reports.

4. Incomplete reports.
5. Reports too late to be of use.
6. Separate reports which should be consolidated.

Changing to a computer system offers the opportunity for a review of existing procedures to eliminate inefficiencies and to incorporate reports not currently available. Interviews with management will in most cases be the source of information regarding the desirability of reports.

The existing system is usually the starting point in assembling the information needed as input and output. Other sources of information are

1. Legal Requirements: In the payroll system certain reports are demanded of employer by the government such as the quarterly FICA and income tax reports, annual W-2 forms and others. The requirements in these areas are mandatory and must be output of the system.

2. Consulting Services: The experience of the manufacturer can be a source of information concerning the general requirements of a data processing system. Management consulting services are generally keeping abreast of computer developments, and some are specializing in electronic data processing.

A computer run converts a particular input to a particular output. The conversion may consist of one or more passes of the data through the computer. A description of some types of runs follows.

INPUT VERIFICATION

The accuracy of output can be no better than the accuracy of the input. Input verification is designed to detect various types of errors in the input.

The first type of error is called implausible. An implausible error stalls the computer, because it is unintelligible to it, and must be detected. An alphabetic in a numeric field is an example of an implausible error. An attempt to add the alphabetic to another would stall the computer. The operator is normally not familiar with the routine and would have no means of correcting the situation. Nor could the operator step the computer past this point. The occurrence of an implausible error stops the system. The routine must be designed to protect itself against implausible errors and write the error items on an error tape.

The second type of error is called "plausible but wrong". A "plausible butwrong" error does not stall the computer but does produce incorrect output. The reporting of 28 hours worked in a day is an example of a "plausible but wrong" error. This type of error item would also be written on an error tape.

A third type of error is called "plausible but probably wrong". The reporting of 12 hours overtime in one day is an example of a "plausible but probably wrong" error. Such an error can be processed and flagged for later inspection by the payroll department.

Input errors can also be studied from the standpoint of their source. The operations performed by the Univac System may be considered the function of an organization called the data processing center. The data processing center is an organization formed to render services to such subscribers as the payroll, purchasing, accounting and engineering departments. Errors in data exist because of introduction by either the data processing center or the subscriber. The center may alter valid data during the transcription of data from document to tape. To minimize such errors unityped tapes are verified on the Verifier.

The detection of input errors caused by improperly prepared source data is the subject of input verification. This run may test the input for

1. alphabetic characters in numeric fields,
2. numeric fields within certain limits,
3. key field validity

and 4. consistency of data.

The validity of keys can be determined by checking for the presence of a correct final digit in the key. Consistency errors are typified by a case such as a medical absence entry in a clock card item also containing a standard work week key.

ITEM REARRANGING

It is occasionally desirable to edit certain kinds of input - put it into a form which will effectively reduce the elapsed time of later runs of the system. In using the Card-to-Tape Converter, for example, not all columns are always utilized. It may be desirable to reduce the 120 digit 10 word item to a 2 word item if between 13 and 24 columns have been used, thereby reducing the volume of tape necessary to hold the information by 80%. The reduced volume of tape reduces the read time in later runs by 80%.

SORTING AND MERGING

Sorting and merging have already been discussed in chapter XII.

PROCESSING

The processing run is a catch-all title for the productive runs of the system. In the processing run the computations are performed, records are formed, files are maintained, and desired output is compiled.

OUTPUT PREPARATION

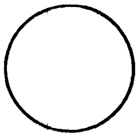
The output units require that data be in a particular format. The final processing run may not record the output in the desired format, because if the processing is complex, editing the output in the main run may take longer than a separate editing run.

In many instances one or more of the operations just described can be done during one pass over the data.

THE PROCESS CHART

The symbols used on a process chart are as follows.

FIGURE 14-1



A single reel of tape. The symbol contains a description of the data on the tape and may represent an input or output file, transactions, errors, etc.

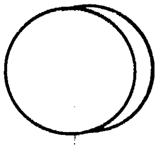


FIGURE 14-2

A multi-reel file. The symbol contains a description of the file.

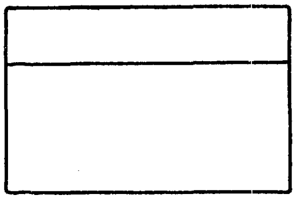


FIGURE 14.3

A run. The symbol contains a run number and a description of the run. The numbering system should be flexible in order that additional runs can be inserted when necessary. One such scheme uses a letter and a number, such as A1.0, A1.1, etc., for each run.

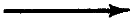


FIGURE 14.4

System flow - shows the flow of data through the system.

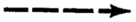


FIGURE 14.5

System cycle - shows that the output of the current system cycle becomes the input of the next cycle.

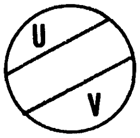


FIGURE 14.6

Unityper conversion of source document to tape, and verification of the unityping.

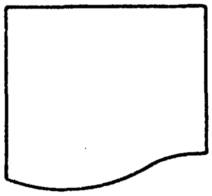


FIGURE 14.7

Printed output from the High-Speed Printer or Supervisory Control Printer.

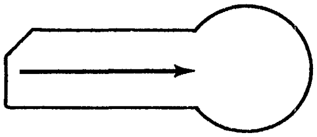


FIGURE 14.8

Card-to-tape conversion.

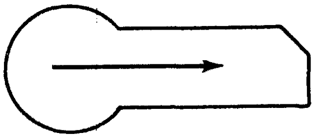


FIGURE 14.9

Tape-to-card conversion.

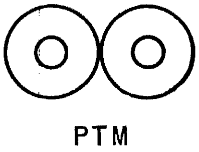


FIGURE 14-10

Paper to magnetic tape conversion.

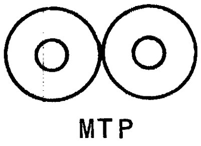


FIGURE 14-11

Magnetic to paper tape conversion.

To continue the reference to the payroll system, it will be recalled that at least two inputs were necessary to supply the output. One contained the variable data such as hours worked; the other was a file of relatively permanent data - the master file.

The master file being relatively permanent infers that occasional changes will be made to its contents. Some possible changes are

1. effecting a transfer of an employee from one section of the master file to another,
2. additions required by hiring new employees,
3. deletions required by the termination of employment

and 4. subsitutions of new information for old.

The effecting of such changes is called file maintenance. The file must be maintained on some cyclical basis: hourly, daily, weekly, biweekly, monthly, etc. The file must be updated at least as often as it is utilized in a processing operation. For example, if payroll is being run weekly, changes to the master file must be administered at least weekly if accurate processing is to be achieved. Even distribution of the work load may require the processing of one fifth of the master file daily.

The master file will be assembled in sequence at the time it is originated. Subsidiary files such as the change file will be prepared on tape and sorted into the same sequence as the master file. Applying the changes becomes a match-merge operation.

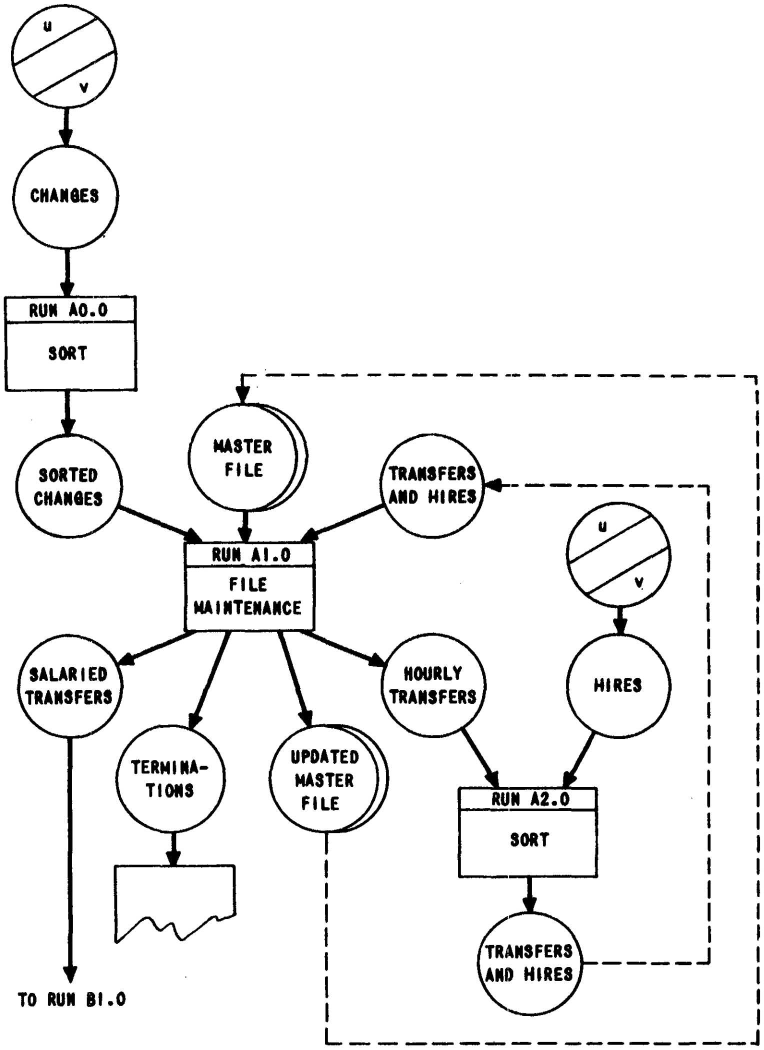


FIGURE 14.12

The following is the process chart of a typical payroll cycle.

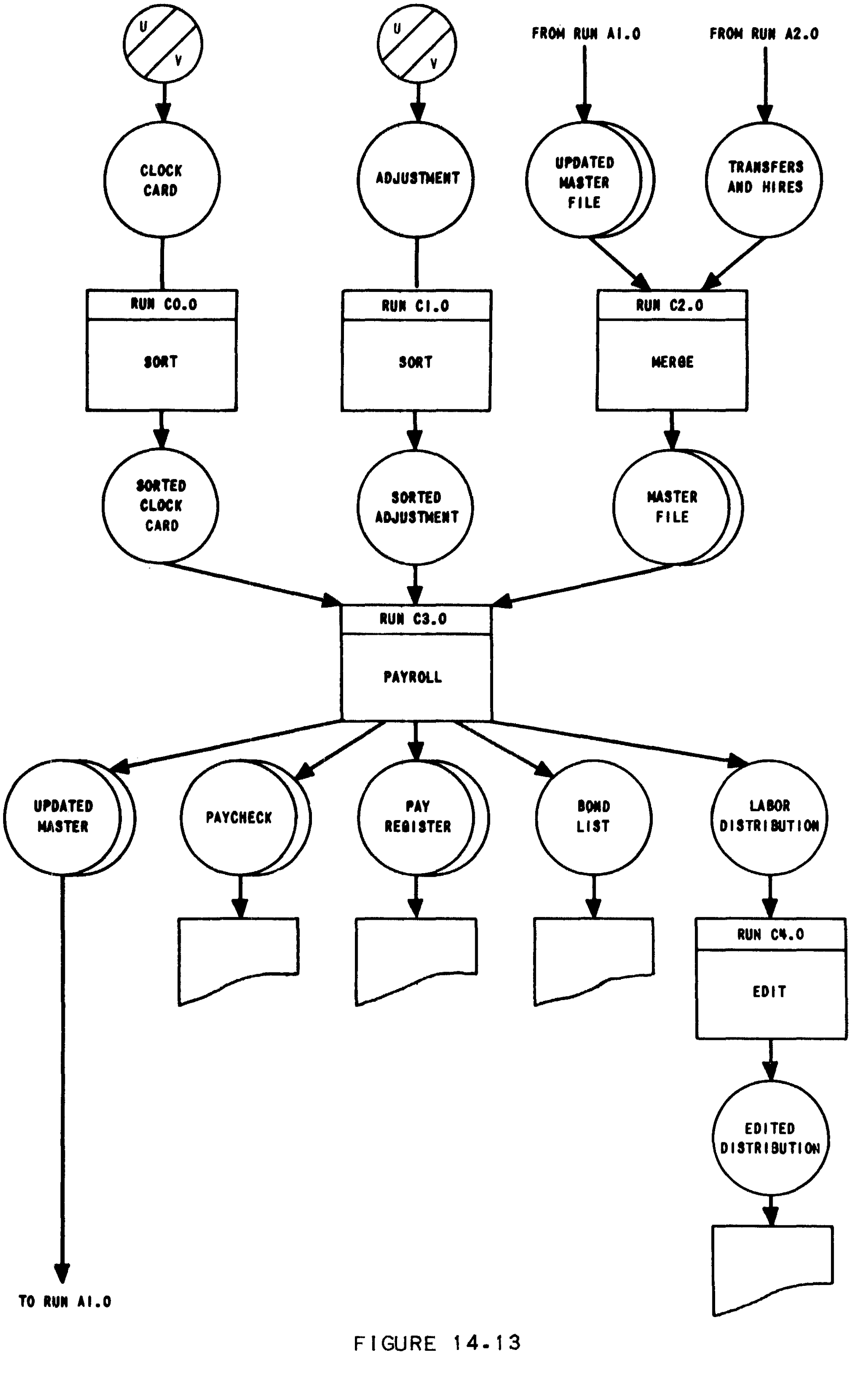


FIGURE 14-13

In run C3.0 a gross pay is computed by multiplying hours worked by hourly rate of pay and then adjusting, if necessary. A net pay is derived by subtracting all deductions from gross pay. This net pay is recorded along with the employee's name, etc., on the paycheck tape. The other outputs are self explanatory. Run C3.0 might use the following servo allocation

2	Paycheck
3	Paycheck Alternate
4	Pay Register
5	Pay Register Alternate
6	Bond List
7	Labor Distribution
8	Sorted Clock Card
9	Sorted Adjustment
–	Master File
A	Master File Alternate
B	Updated Master File
C	Updated Master File Alternate

The run might also use the following memory allocation for input-output blocks.

1520	1579	Master File
1580	1639	Sorted Adjustment
1640	1699	Sorted Clock Card
1700	1759	Labor Distribution
1760	1819	Bond List
1820	1879	Pay Register
1880	1939	Paycheck
1940	1999	Updated Master File

The number of available Uniservos and the memory capacity will determine the amount of processing that can be done in one run. If the number of Uniservos available is exceeded by the requirements of the input and output, or if the instructions exceed the memory capacity, the run may be sub-divided into two runs with an intermediate output.

A run is said to be tape limited when the reading or writing of data takes longer than the processing of the data. A run is said to be computer limited when the processing takes longer than the reading or writing.

In applying the computer to a data processing system it is economically desirable to keep the time spent on the computer to a minimum. In general, the programmer will be faced with this antithesis: fewer iterative routines will decrease running time but use more memory space, more iteration will cost more in time but use less memory space. The nature of the data processing system may dictate the processing to be done in a run. The programmer's job is to (1) do the processing, (2) in the shortest period of time and (3) keep the time versus memory relation in balance.

TIME ESTIMATION

COMPUTER RUNNING TIME

Routine timing has already been discussed on pages196-212.In addition to running time about one half minute must be allowed for each tape mounted while the computer is not processing other data. Moreover, rewind time must be allowed for each group of Uniservos rewinding in parallel while the computer is not processing. By allocating two Uniservos to every file consisting of more than one tape, mounting time can be eliminated for all tapes but the first, and rewind time, for all tapes but the last. By a judicious allocation of Uniservos between runs it is often possible to eliminate even this mounting and rewind time.

UNITYPING

Approximately 13 blocks can be unityped per hour.

VERIFYING

Verifying time is approximately the same as unityping time.

CARD-TO-TAPE

The Card-to-Tape Converter converts 240 cards per minute. For each tape converted rewind time plus about one half minute for set up time must be allowed.

UNIVAC HIGH-SPEED PRINTER

The High-Speed Printer prints 600 lines per minute and fast feeds at 20 inches per second. For each tape printed rewind time plus about one half minute for set up must be allowed.

APPROXIMATE TIME ESTIMATING

It is sometimes convenient to ascertain the approximate running time of a run without going into the detailed timing of the coding. This approximation will only be as accurate as the estimate of the processing time but may aid in the choice of one of several possible approaches to a system.

The following steps are required to make an approximate time estimate.

1. Determine the number of blocks written or to be written on each reel in the system.

2. Preparation of Input: the time needed for unityping, verifying, etc., is calculated as explained above.

3. Estimation of processing time - this is usually an estimate to produce one block of output from one block of input. If there is little processing and/or the item size is large, assume that the operation is tape limited. If there is much processing and/or the item size is small, assume that there operation is computer limited. Thus the processing time may be assumed to be tape time, etc.

4. Overall Processing Time: because reading, writing, and processing take place simultaneously, the overall processing time is determined either by the amount of input or the amount of output, whichever is larger. The number of blocks, as determined in 1, is then multiplied by the processing time assumed in 3.

5. To this add the input preparation time, from 2, and the time required for the High-Speed Printer, or any output unit used.

TAPE USE	SPACING IN INCHES	CHARACTERS PER INCH	TYPE OF TAPE	BLOCKS PER REEL (*BLOCKETTES)	READ-WRITE TIME		MINUTES REWIND PER REEL	FEET UTILIZED PER REEL
					MILLISECONDS PER BLOCK	MINUTES PER REEL		
COMPUTER	3.6/block 1 between blocks	200	Metallic	4000	54.0	3.60	3.07	1533
			Mylar	6200		5.58	4.75	2377
	2.88/block 1 between blocks	250	Metallic	4,700	43.8	3.43	3.04	1520
			Mylar	7,400		5.4	4.79	2393
UNITYPER II	2.4 / blockette 2.4 between blockettes 2.4 between blocks	50	Metallic	500*	293.0	24.41 sec	24 sec.	200
CARD-TO-TAPE CONVERTER	.9375/blockette 1.8 between blockettes 2.4 between blocks	128	Metallic	6,400*	175.25	3.12	3.03	1513
			Mylar	10,000*		4.83	4.73	2365
TAPE-TO-CARD CONVERTER	.9375/blockette .1 between blockettes 2.4 between blocks	128	Metallic	12,900*	90.25	3.23	3.06	1527
			Mylar	20,000*		5.01	4.74	2368
HIGH SPEED PRINTER	.9375/blockette 1.0 between blockettes 2.4 between blocks	128	Metallic	8,400*	135.25	3.16	3.04	1520
			Mylar	13,200*		4.96	4.78	2388

PLANNING THE CUT-OVER

When the routines have been prepared, some method must be devised for converting from the present system to the computer system. In preparation for this cutover the master data must be converted from its present form to tape files. Once this conversion has started some provision must be made for keeping a record of every change of the data that occurs after the time of conversion. This change record will be used to bring the tape files up to date when use of the computer system actually begins.

Another problem to be faced in the cutover period is the retraining of personnel concerned with the application. Those persons concerned with the origination of data for the system must be familiarized with the forms to be used in introducing data into the system, their proper completion and their channelling to the computer area. Those people concerned with the use of data provided by the system must become informed of the information they can expect from the system, the form in which it will come and its frequency.

A third area of concern in the cutover period is the debugging of the routines, which then leads to the conversion of the present system to the Univac system.

It is not usually possible to convert from one system to another merely by halting operations on the old system and starting on the new. There is characteristically a period of time during the conversion when the two systems are run in parallel. The parallel operation is usually entered gradually. Thus, a billing operation might be cut over one district at a time. In this way the initial cutover provides an opportunity for trial runs of the routines against real data. The results of these trial runs must be verified in complete detail.

The duration of parallel operation depends on various factors, among them being the degree of success with which the routines execute the trial runs and the variability possible in the input data. The parallel operation should be rather firmly scheduled in order that it run as long as necessary for assurance of accuracy in the routines but no longer than necessary, since parallel operation is, by itself, an uneconomical procedure. It is customary to speed up the rate of cutover as the cutover period proceeds.

FUTURE PLANNING

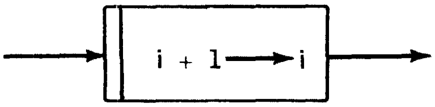
The initial applications of a computer have as one of their purposes the defraying of the cost of the computer. In the usual case, this is not the end but only the beginning of the use of the computer. To put the computer to its fullest use key personnel throughout the company should become acquainted with the capabilities of the computer. Then, valid conclusions can be drawn as to the operations to be prepared for the computer and the priority to set up for the chosen operations. One of the better methods for acquainting personnel with the computer appears to be a series of company seminars given to department heads and supervisors.

In deciding on the initial applications of a computer the usual consideration is savings of cost in the specific sense. The question asked is, "How much can be saved by putting this operation on the computer?". Savings in cost is always the basic consideration in deciding on a computer application, but in later applications, this savings may be conceived of in a more general sense. Instead of asking specifically about savings in cost, the question brought forward might be, "If we put this operation on the computer, can we get the information we want in a better form than we do now?" or "Can we get information that we need but which was never before available?" or "Can we realize a significant savings in time?" It is criteria such as suggested by the above questions that determine the feasibility of an application. Some possible applications of this nature are scientific inventory and production control, operations research, market analysis using census data, and determination of product mix. Another field that should not be overlooked is the improvement of already existing routines, thus taking advantage of the experience garnered from the initial preparation of these routines.

chapter 15

Operational Routines

In this chapter there will be described solutions to problems frequently encountered in using a computer as a data processor. Each solution will be shown in an abbreviated flow chart. The operations making the next input item available or of recording on tape the current output item will be indicated by a double-lined box adding 1 to a letter subscript:



This symbol will stand for all operations implied by selecting the next item of a block. This includes getting the next block when the current one is exhausted, or the next tape when the present one is completely read. A similar symbol will represent the appropriate output operations.

TAPE SUMMARY

A frequent problem encountered in computer applications is to print a summarization of a detail tape. To illustrate the problem and its solution, a practical example will be given. Consider a file of insurance policies, each policy represented in the file by an item, P_i, containing at least the following fields:

1. The insured's occupation classification code, P_i^c
2. The age of the insured at the time of issuing the policy, p_i^a
3. Type of insurance issued (the plan), P_i^p
4. The amount of insurance purchased (face value), P_i^f

A table is to be produced, similar to the one illustrated in Figure 15-2,showing a summary of the total amount of insurance and number of policies, by type of insurance, by age at issue, and by occupation of insured.

Of course, not all occupations, nor all ages, nor all plans may be contained in this file. Further, assume that the total combinations of occupation, age and plan exceed the memory capacity of the computer.

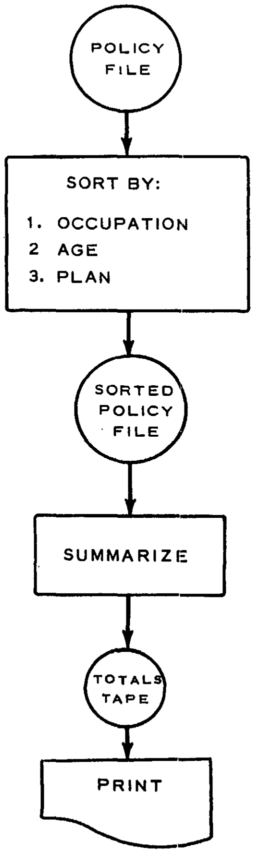


FIGURE 15-1

OCCUPATION CODE	AGE AT ISSUE	PLAN	AMOUNT INSURED	NUMBER OF POLICIES
		A	1,230,000	850
		B	2,000,000	501
		C	1,600,000	350
	25		4,830,000	1701
		A	2,000,000	900
		B	650,000	100
		C	15,050,000	1500
		D	205,000	73
	30		17,905,000	2573
401			22,735,000	4274
		A	6,365,000	1055
		C	6,160,000	1231
	27		12,525,000	2286
		A	3,121,000	630
		G	8,900,000	2461
	28		12,021,000	3091
		A	4,221,000	1347
	29		4,221,000	1347
435			28,767,000	6724
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.

FIGURE 15-2

The main steps in the solution are shown in Figure 15-1. The first operation is to sort the policy file into an ascending sequence in order by, from major to minor, occupation, age and plan. This is accomplished by one of the standard sorting routines which were discussed in Chapter XII. The output of the sort is the sorted policy file which forms the input to the next operation which is the summarizing run. Figure 15-3 represents the essential steps in this summarization.

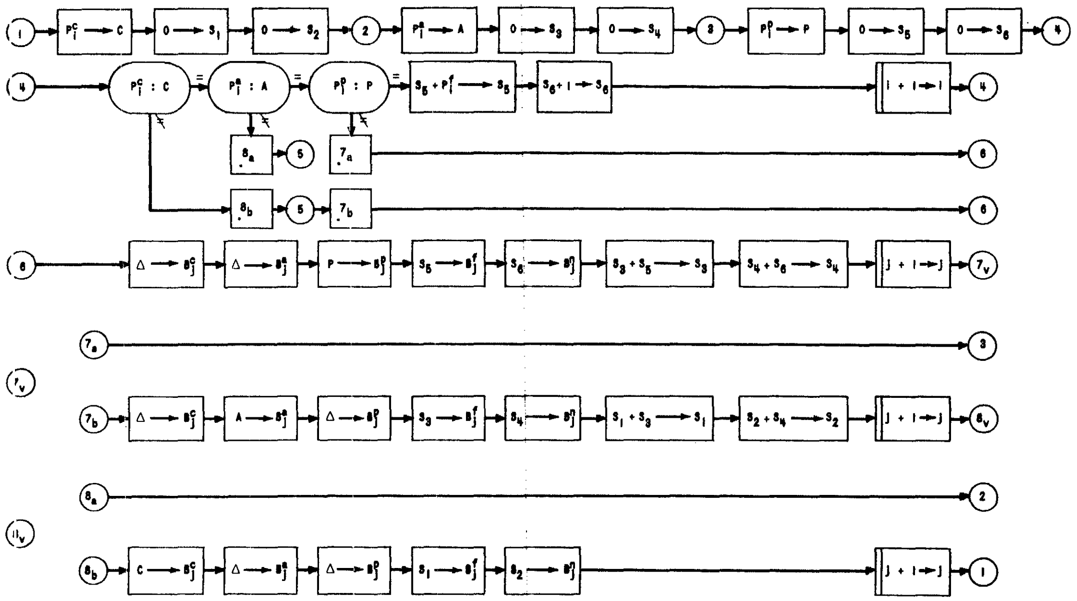


FIGURE 15-3

Since the policy tape has been sorted, the policy items with given occupation, plan and age will be adjacent to each other on the tape. The first operation, from ① to ④, is to store the occupation code, age and plan fields of the first policy item. In addition, six tallys are set to zero which will be used in accumulating the total face value and number of policies issued for each classification. At ④ the occupation code and age and plan of the ith policy item are compared with the occupation code and age and plan stored. They must agree, and the face value of the policy is added to S_5, and one is added to S_6, which is the count of the number of policies issued with classification CAP. The next policy item is selected, and control is transferred to ④ to process this item.

The keys of the record item and the ones following are compared in turn to the keys stored. When a change of key occurs, control is transferred to the output routine (connectors ⑥ to $\textcircled{8v}$).

Each output item, B_j, consists of the following five fields:

B_j^c = The occupation code field

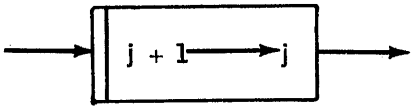
B_j^a = The age field

B_j^p = The type of insurance field

B_j^f = The accumulated face value of policies with keys CAP

B_j^n = The total numbers of policies with keys CAP

When an item with a different plan key is found, $\textcircled{7a}$ is set and control is transferred to ⑥. Non-printing characters (space symbols, Δ) are inserted into B_j^c and B_j^a. The plan, P, is inserted into B_j^p, and S_5 and S_6, the totals, are inserted respectively into B_j^f and B_j^n. The next operation is the addition of S_5 to S_3 and S_6 to S_4 (sub-totals for occupation code C and age A), since the plan, P, has changed. The box



implies all operations necessary to place B_j on the output tape. Connector $\textcircled{7a}$ was set and, therefore, control is transferred to ③ where S_5 and S_6 are reset to zero in preparation for totaling the next plan. In addition, the new plan is stored in P, and control is transferred to ④.

When an item is found which contains a new age key, connectors $\textcircled{7b}$ and $\textcircled{8a}$ are set, and control is again transferred to $\textcircled{6}$ where an output item containing the totals under P is formed as previously. In this case, since $\textcircled{7b}$ is set, an output item containing the age totals in formed. Δ's are inserted into B^c_j and B^p_j. The age, A, is inserted into B^a_j, and the totals under classification CA, S_3 and S_4, are inserted into B^f_j and B^n_j respectively. Then the age totals are added to S_1 and S_2 (the totals for class C). Connector $\textcircled{8a}$ then transfers control to $\textcircled{2}$ where S_3, S_4, S_5, and S_6 are reset to zero, and the new age and plan are inserted into A and P.

When an item with a new occupation code is found, connectors $\textcircled{7b}$ and $\textcircled{8b}$ are set, and control is transferred to $\textcircled{6}$ where, as shown previously, output items for the plan and age totals are formed. Then $\textcircled{8b}$ causes an output item to be formed for the totals under the classification code C. C is inserted into B^c_j; and Δ's, into B^a_j and B^p_j. S_1 and S_2 are inserted into B^f_j and B^n_j respectively; and the output operations, executed. Control is then transferred to $\textcircled{1}$ where all the totals S_1 to S_6 are reset to zero, and the new occupation code, age, and plan are inserted into C, A, and P, respectively.

The reader will note that at any time a new policy item is selected for a different plan, age or occupation code the totals to date are placed in an output item B_j, and the totals for this category and its subcategories are reset.

The output of this summary run, then, consists of the items B_j which represent the totals for each CAP. Printing this tape produces the table of Figure 15-2.

If it is desired to have the table list the summaries in the order: 1. occupation code, 2. age, and 3. plan, the procedure should be modified in this fashion: Since the output items representing the totals for the major categories follow the items with summaries for the minor categories, each completed output tape, instead of simply being rewound when it has been filled, should be read backwards, its items being written on a new output tape exactly in the order they are read .

Thus, this second output tape now contains the major totals first, then, the minor totals. The last reel of tape coming from the summary run should be the first one printed, then the next to the last tape should be printed, etc. Of course, this would give a table arranged in descending sequence. To avoid this, the sort routine should produce a descending sequence rather than an ascending one. The summary run itself is not changed.

TABLE LOOK-UP

Many data processing problems involve "Table Look-Up" operations. That is, given a quantity x, select from among a set of quantities Y a quantity y which is assigned to x. Wherever possible, keep the size of the table Y as small as possible. In some cases it may be possible to reduce the table to a formula from which y can be computed, given x. However, in some applications it is not possible to reduce the table to a size which can be stored in the memory or to a formula. In these cases, it is necessary to consider table look-up solutions that are completely general as far as table size and argument interval are concerned.

Consider the following problem. A file contains a series of billing items, B_i, containing among other things, the following fields:

1. Location code of point of origin from which item purchased was shipped, B_i^o
2. Destination code where item was shipped to, B_i^d
3. Commodity classification of item, B_i^c

It is desired to obtain the shipping rate by looking this rate up in a table which is entered by origin code, destination code, and commodity classification code.

Consider the table to be a file consisting of items T_j containing the following fields:

1. Point of origin code, T_j^o
2. Point of destination code, T_j^d
3. Commodity classification code, T_j^c
4. Rate for this origin, destination, and commodity, T_j^r

The file of items T_j which constitute the table are assumed to be arranged in an ascending sequence, from major to minor, by origin, destination and commodity. This arrangement is effected once, and once only, at the time the table is developed.

The main steps in the table look-up are shown in Figure 15-4.

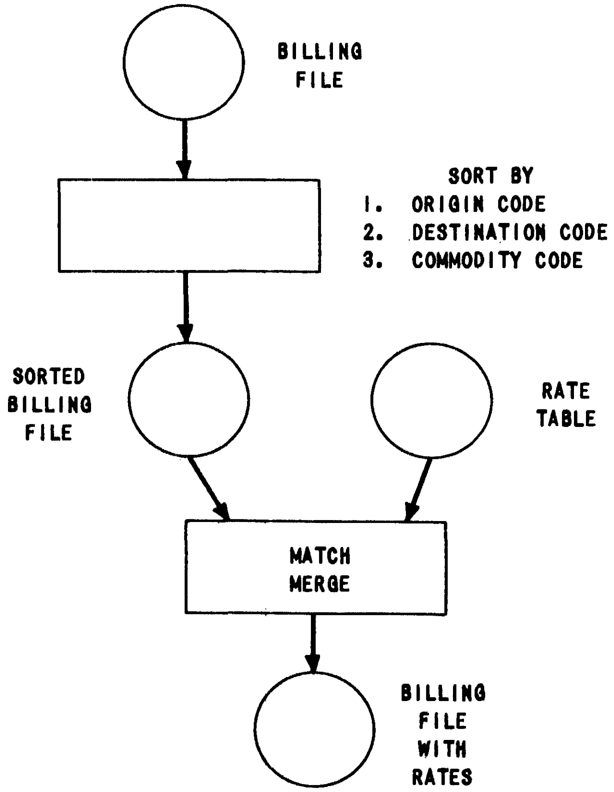


FIGURE 15.4

The first operation is to sort the billing items B_i into an ascending sequence, from major to minor, by origin code, destination code and commodity classification. This is accomplished by a standard sort routine. The next operation is to match merge the sorted billing file and the table, thus producing an output which consists of the same billing items with the appropriate rate inserted in them.

The essential elements of the match merge operation are shown in the flow chart which is Figure 15-5.

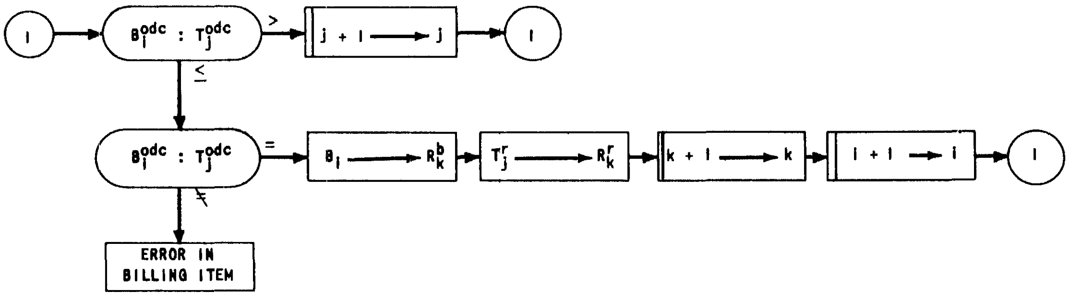

1
$B_i^{odc} : T_j^{odc}$
>
$j + 1 \longrightarrow j$
1
≤
$B_i^{odc} : T_j^{odc}$
=
$B_i \longrightarrow R_k^b$
$T_j^r \longrightarrow R_k^r$
$k + 1 \longrightarrow k$
$i + 1 \longrightarrow i$
1
≠
ERROR IN BILLING ITEM

FIGURE 15.5

The table items T_j are examined successively until an item with origin, destination and commodity code is encountered which matches those codes in the current billing item B_i. When the match occurs, an output item R_k is formed by attaching to the billing item the rate field of the table item, T_j^r. The box $\rightarrow\boxed{k+1 \rightarrow k}\rightarrow$ implies the output operations necessary to record W_k on tape, while the box $\rightarrow\boxed{i+1 \rightarrow i}\rightarrow$ selects the next billing for the table look-up.

In some applications the table look-up operation may involve an interpolation between near lying entries in the table. In this case, while the general procedures shown in Figure 15-4 are unchanged, a modification of the match merge operation is needed.

Assume that the billing item, B_i, contains an argument, B_i^a, which is the basis of of the table look-up(this corresponds in function to the fields B_i^o, B_i^d, and B_i^c, of the previous example). Suppose further that four point interpolation is needed in selecting the rate. That is, if the symbol E_n^a represents the argument of the nth table entry, then if

$$E_{n-1}^a < B_i^a \leq E_n^a$$

the table values for arguments E_{n-2}^a, E_{n-1}^a, $E_n^{a'}$, and E_{n+1}^a will be needed. The mathematical formula using these entries and their arguments to calculate the interpolated rate will be indicated by $F(E_{n-2}, E_{n-1}, E_n, E_{n+1})$.

The flow chart shown in Figure 15-6 is the required match merge necessary to select the required table entries E_n noted above.

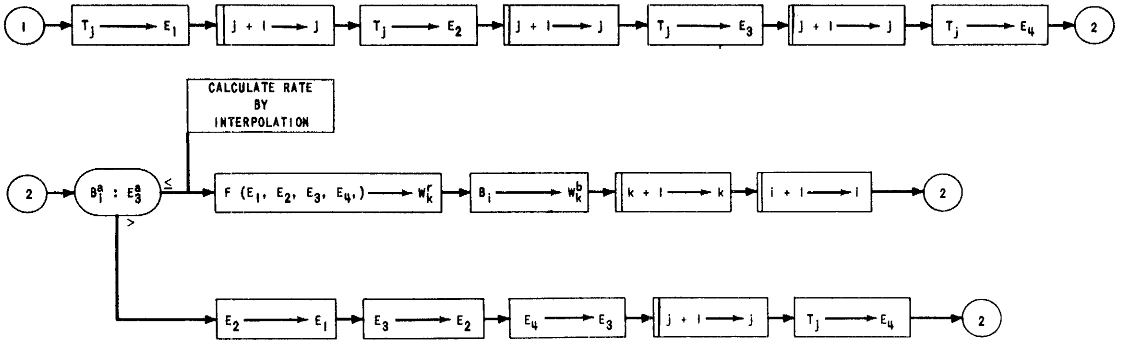


FIGURE 15-6

The first two entries of the table must correspond to arguments below the range of arguments B_i^a. Similarly, the last two entries of the table must correspond to arguments above the range of B^a_i. The initial operations, performed once only, are ① to ② . These steps stored the first four entries and their arguments (T_1, T_2, T_3 and T_4) as items E_1, E_2, E_3 and E_4, respectively. At ② the table look-up begins. The first billing item argument B^a_i is compared with the argument E^a_3. If B^a_i is greater, each En is displaced down one position with E_1 being dropped and the next Tj becoming E_4. When, finally, the first E^a_3 is located which is just greater than (or equal to) B^a_i, the four items E_1, E_2, E_3, and E_4 contain the proper entries for interpolation. An output item W_k is formed consisting of the billing item and the interpolated rate. This item is sent through the output operation $\rightarrow \boxed{k+1 \rightarrow k} \rightarrow$ necessary to record it on tape, and the next billing item selected.

The extension of this flow chart to handle 2, 3, 5 point, or higher interpolation is obvious.

EXPLOSION CALCULATION

The explosion calculation can be described by the following problem. A company manufactures a number of models of a product. For each model a bill of materials exists which lists the basic sub-assemblies or units and the number required for each model. This data can be termed a bill of materials file consisting of items M_i. Each item represents a unit or sub-assembly for a particular model. It contains, among other things, the following fields:

1. The model code to which this unit belongs, M^m_i
2. The part number of a part used on this model, M^{pn}_i
3. The number of such parts used on this model, M^n_i

This bill of materials file is kept in model code sequence to facilitate the problems of file maintenance and the explosion run to be described.

A second file, the production schedule, is also available. This file consists of a series of items, P_j, containing the following fields:

1. The model code, P^m_j
2. The number of such models to be constructed, P^n_j

The problem is to determine the total number of sub-assemblies required by the production schedule. That is, the production schedule is to be "exploded" into the pieces that make up the models.

Figure 15-7 depicts the major operations required in exploding the production schedule.

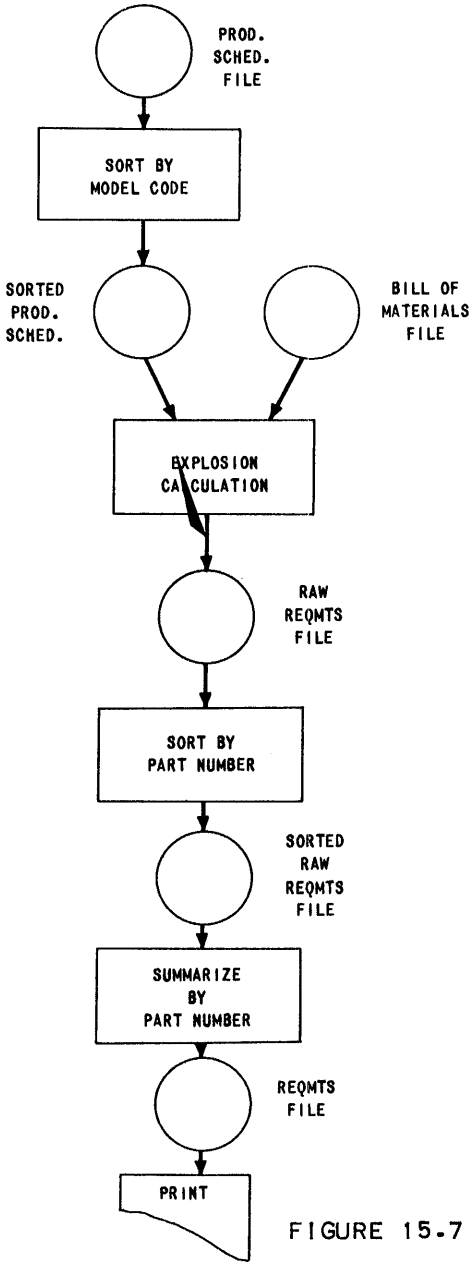


FIGURE 15.7

Assuming a random development of the production schedule, the first step is to sort this schedule into model code order to facilitate its "multiplication" by the bill of materials. This is accomplished through one of the standard sorting routines.

The output of this run is called the sorted production schedule which forms with the bill of materials file the input to the explosion calculation. In this operation, the number of units or sub-assemblies required to produce the quantity of each model listed on the production schedule is determined. The output of this calculation is calledthe raw requirements file. Now, because many models contain common sub-assemblies, it is necessary to summarize the raw requirements file.

First, of course, the file must be sorted to part number sequence not only for the summary to follow but also for the convenience in reading the printed sub-assembly requirements table. The summarization operation has already been described.

Figure 15-8 is a flow chart showing the method of calculating the raw requirements.

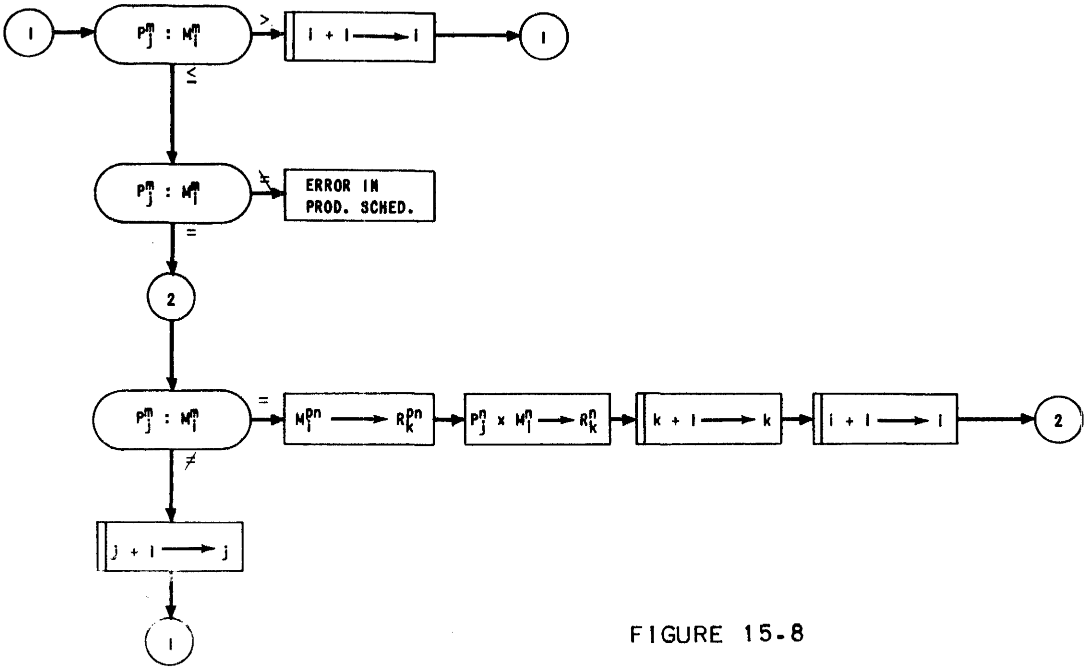

1
$P_j^m : M_i^m$
>
$i + 1 \longrightarrow i$
1
≤
$P_j^m : M_i^m$
≠
ERROR IN
PROD. SCHED.
=
2
$P_j^m : M_i^m$
=
$M_i^{pn} \longrightarrow R_k^{pn}$
$P_j^n \times M_i^n \longrightarrow R_k^n$
$k + 1 \longrightarrow k$
$i + 1 \longrightarrow i$
2
≠
$j + 1 \longrightarrow j$
1

FIGURE 15-8

At (1) the model codes of the first production schedule item andthe first bill of materials item are compared. If the model code calledfor by the production schedule is the larger, it means that its corresponding bill of material items are further up the bill of materials file. Accordingly, this file is advanced item by item until a model code is reached equal to (or less than) the production schedule model code. Next a test is made to detect improper model codes which may have slipped in during the manual operations used in preparing the production schedule. Next, an output item R_k is built up. The part number of the current bill of materials item is stored in R_k^{pn}, and then the number of such parts needed is calculated by multiplying the number of the model to be built, P_j^n, by the number of this part used in that model, M_i^n. This field is the requirement for this part by the production for this model and is designated R_k^n. The box $\rightarrow\boxed{k+1 \rightarrow k}\rightarrow$ carries out the steps necessary to record this requirement item on tape. The bill of materials file is then advanced one item and this item's model code checked against the current production schedule item's model code. If they agree, another extension is made. This process continues until a bill of materials item for a different model code turns up. This signifies that all of the extensions for current production schedule item's model code have been made, and the production schedule file is then advanced one item.

Having seen how a simple explosion run is performed, consider a somewhat more involved and, thus, more practical problem. Suppose that our production schedule consists of a series of items giving the required production per month, per model for a certain number of months. That is, each production schedule item, P_j, contains the following fields:

1. Model Code, P_j^m
2. Number of units to be produced this month, P_j^n
3. Coded representation of this month, P_j^d

Further, suppose that if a model is to be produced for a given month, each of the sub-assemblies will have a lead time peculiar to the assembly unit. For example, if a model is to be completed on day X, sub unit A must be available on day X-L, or L days earlier. Thus, modify the bill of materials file so that it includes the appropriate lead time. Each bill of materials item will now contain the fields:

1. The model code to which this unit belongs, M_i^m
2. The part number of this unit, M_i^{pn}
3. The number of such units used on this model, M_i^n
4. The amount of lead time required for this unit, M_i^l

Now compute the "phased" requirements. That is, determine not only what and how many sub-assemblies are required for this production schedule, but also on what date they are required. Figure 15-9 shows the general sequence of steps required in calculating the phased requirements.

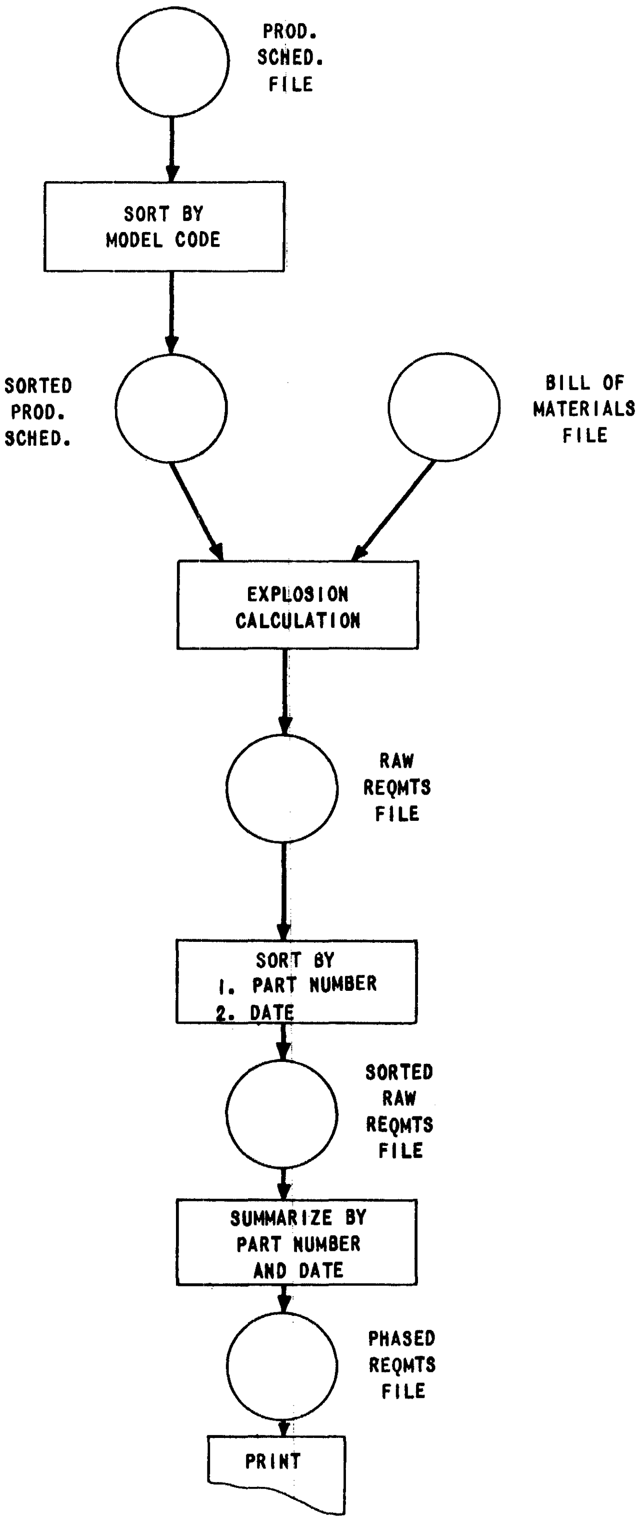

PROD.
SCHED.
FILE
SORT BY
MODEL CODE
SORTED
PROD.
SCHED.
BILL OF
MATERIALS
FILE
EXPLOSION
CALCULATION
RAW
REQMTS
FILE
SORT BY
1. PART NUMBER
2. DATE
SORTED
RAW
REQMTS
FILE
SUMMARIZE BY
PART NUMBER
AND DATE
PHASED
REQMTS
FILE
PRINT

FIGURE 15.9

The same essential steps are found in this solution as described earlier for Figure 15-7. Of course, the explosion calculation will necessarily be different. The flow chart of this explosion run is shown in Figure 15-10.

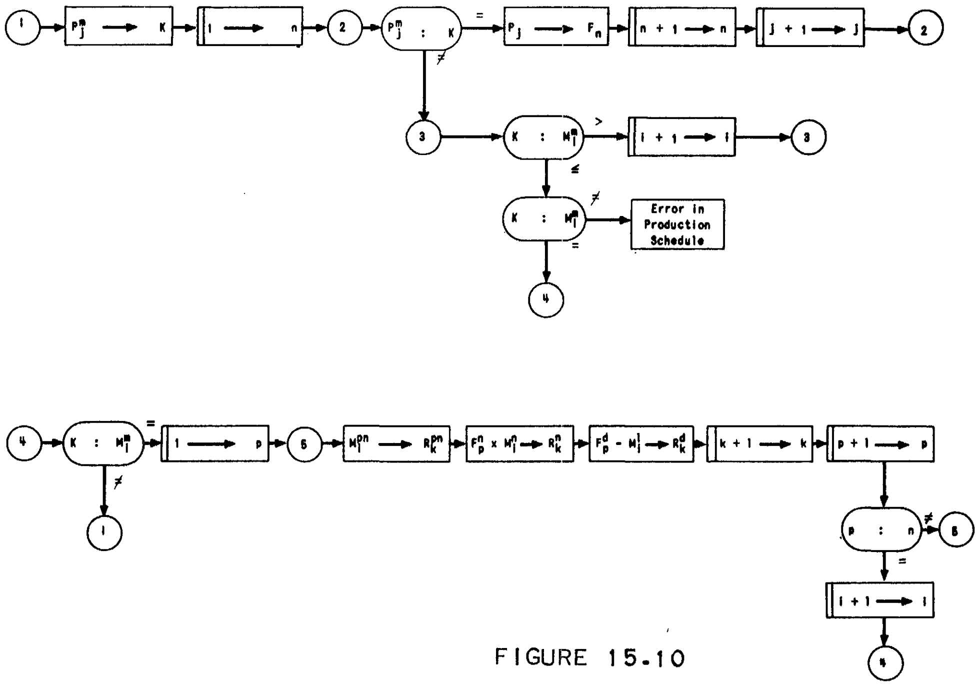


FIGURE 15.10

At ① the model code of the first production schedule item is stored as a key K. Beginning at ② each production schedule item with the same model code K is stored in the memory. These stored production items are called F, any particular one being F_n. As soon as a production item is found for a different model code, go to ③ where the bill of materials file is advanced to the first item for model code K.

Beginning at ④ start exploding the production schedule. The first stored production item, F_p with $p=1$, is selected and the number of units to be produced during month D is multiplied by the number of sub-assemblies M_i^n required. Then, the lead time, M_i^l is subtracted from the completion date, F_p^d, and these two fields and the stock number of the sub-assembly are placed in an output item, R_k. The box $\rightarrow\boxed{k+1 \rightarrow k}\rightarrow$ implies the output operations necessary to record the item R_k on the raw requirements tape. The box $\rightarrow\boxed{p+1 \rightarrow p}\rightarrow$ selects the next stored production item, and it is processed in a similar fashion. When all of the stored production items, F_p, have been extended the bill of materials file is advanced to the next sub-assembly for this model and the process repeated. When all sub-assemblies for model K have been processed this entire procedure beginning at ① is repeated for the next scheduled model.

chapter 16

Storage of Information

Consider the following circuit.

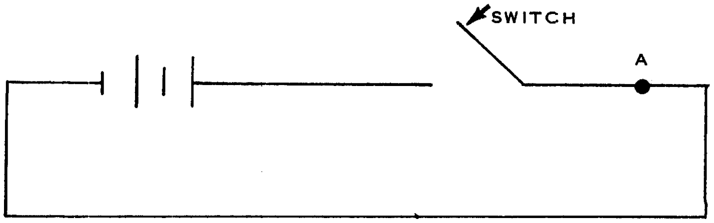


FIGURE 16-1

When the switch is closed a voltage is produced at point A; when open, no voltage is produced. Suppose the switch is operated once every μs in the following way - open, open, close, open, close, open, close - and the voltage at point A is plotted.

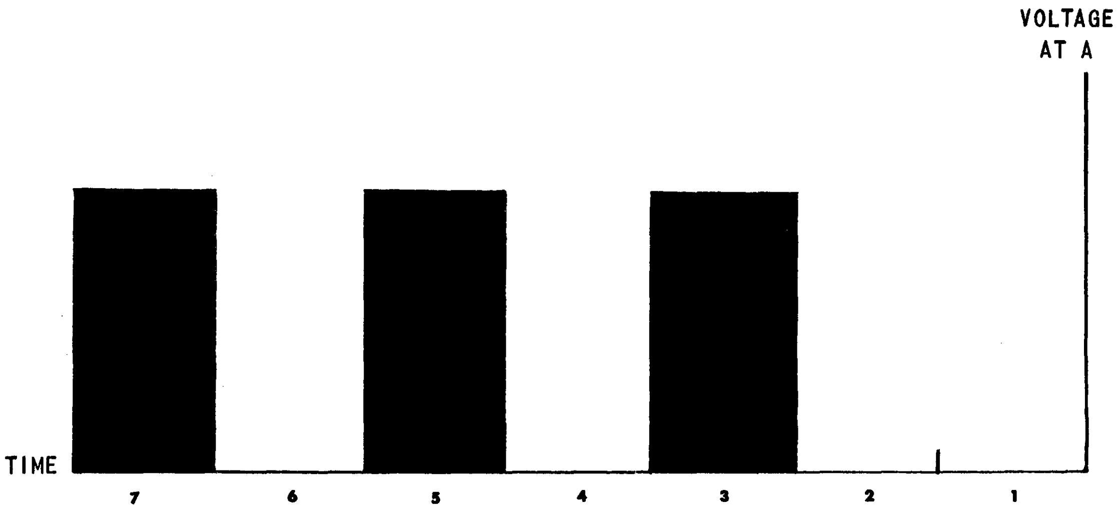


FIGURE 16.2

If voltage represents a one; and no voltage, zero; the operation of the switch sent the representation of the character A past point A. Voltage patterns, or pulse combinations, on a time scale is the method of transferring data in the computer.

The representation of a character is called a binary representation, since the representation consists of a permutation of two digits, one and zero. A one or a zero in a binary representation is called a bit, the name given to a binary digit. Thus, a specific character consists of seven bits, or in general, a character consists of seven bit positions. In the computer, the duration of a voltage level, or pulse, to represent one bit position is .4μs rather than one μs.

MERCURY TANK

Consider the following audio circuit.

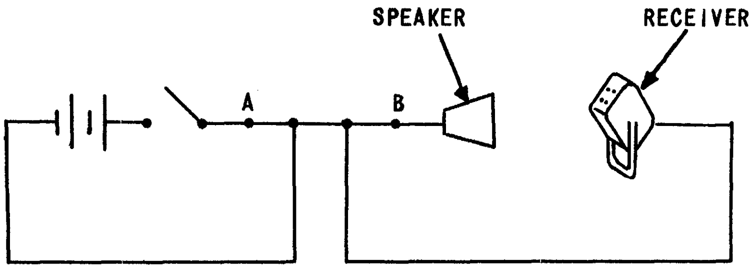


FIGURE 16-3

If the switch were operated as before, the pulse combination will be the same at point B as at point A during the first seven microseconds. If the switch is then left open, the following will occur. When a voltage level, or pulse, reaches the speaker, it will vibrate a thin slab of crystal similar to that in the earpiece of a telephone. The vibration creates a sound wave that travels through the air until it reaches the receiver. Suppose the distance between the speaker and receiver is such that it takes seven μs for the sound wave to cover this distance. When the sound wave reaches the receiver, it vibrates a crystal in the receiver, thus converting the sound wave back into a pulse. Compared to the speed of the sound wave, it can be assumed that it takes no time for the pulse to travel from the receiver to the speaker. With this circuitry, the pulse combination of the character A will be recreated at point B every seven microseconds. The air between the speaker and the receiver has stored the pulse combination. This fact is the storage principle of the mercury tank, except that for echo and power considerations, mercury, rather than air, is used to propagate the sound wave. Register A, rX, rL, rF, CC and CR use mercury tanks as their storage medium.

MAGNETIC CORE

A piece of magnetizable material is thought of as consisting of elementary bar magnets, called domains. The material has two possible states. The domains can be arranged randomly, in which case the material is not magnetized; or the north poles of all the domains can point in one direction, in which case the material is magnetized, or polarized.

If a wire is coiled about a piece of magnetizable material, the following relation holds between the current passing through the wire and the polarity of the material.

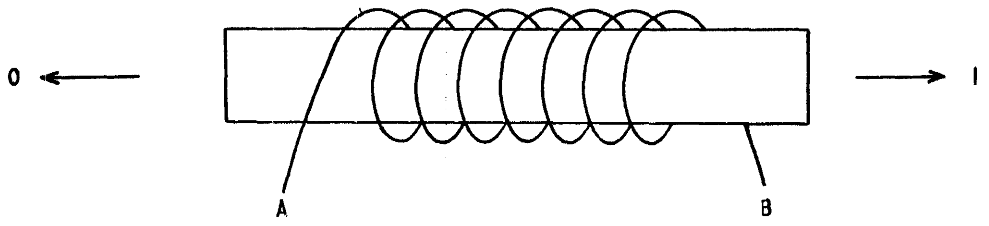


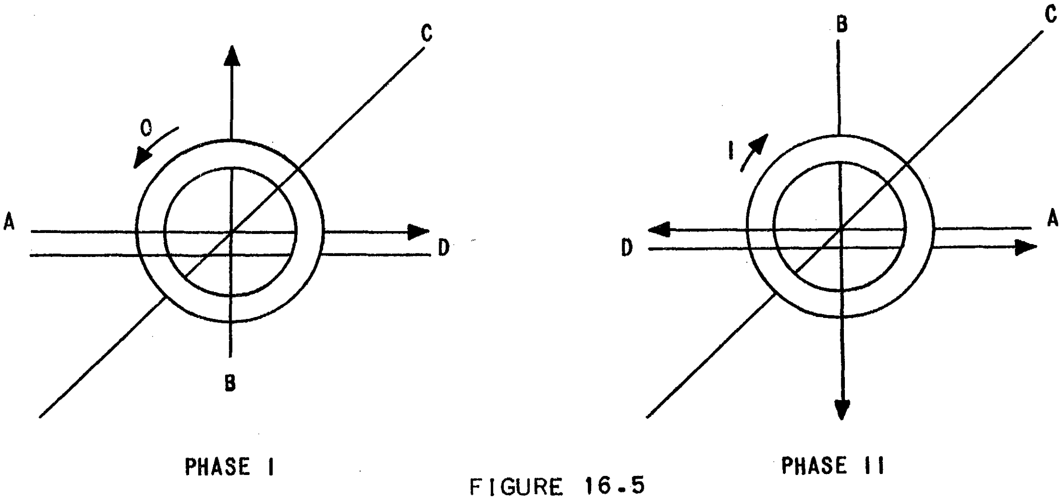
FIGURE 16-4

If sufficient current is passing through the wire from A to B, the north poles of the domains will point to the left, and the material can be said to be in a zero state. If sufficient current is passing from B to A, the north poles will point to the right, and the material can be said to be in a one state.

The above relationship has the following properties. Once the material is placed in a state by a flow of current, it remains in that state after the current has been removed. Also, a characteristic amount of current must be flowing through the wire before the material will change its state. When the material goes from one state to the other, the material is said to flip. For purposes of discussion, say that it takes one jolt of current to flip the material.

Since the zero and one states of the piece of magnetizable material present two easily distinguishable stable states, the material can be used to store one bit position of information. In practice, the piece of material is in the form of a doughnut and is called a core. Instead of being coiled about the core, the wire is passed through the center of the core, the wire thus having a half turn about the core.

Storing information in a core and reading information from a core occurs in two phases, phase I and phase II.



PHASE I

PHASE II

FIGURE 16.5

During phase I a half jolt of current is always sent along each of wires A and B in the directions shown in Figure 16-5. Phase II is the same as phase I except that the direction of the current is reversed.

STORING INFORMATION IN A CORE

During phase I the core is placed in the zero state. If a one is to be stored in the core, during phase II the core is flipped to the one state. If a zero is to be stored, during phase II a half jolt is sent along wire D. This current cancels the effect of the half jolt of current flowing along wire A. The half jolt of current on wire B is not sufficient to flip the core, and it remains in the zero state.

READING INFORMATION FROM A CORE

Just as current on a wire passing through a core flips the core, if a core is flipped, a current is produced on the wire.

If the core is in the one state, the core will flip during phase I, thus producing current on wire C. If the core is in the zero state, the core will not flip during phase I, and no current will be produced on wire C. Only if no current is on wire C during phase I is a half jolt of current sent along wire D during phase II. Thus, phase II restores the core to its original state.

THE MEMORY

Figure 16-6 shows a four cell memory, each cell having the capacity to store one two bit position word.

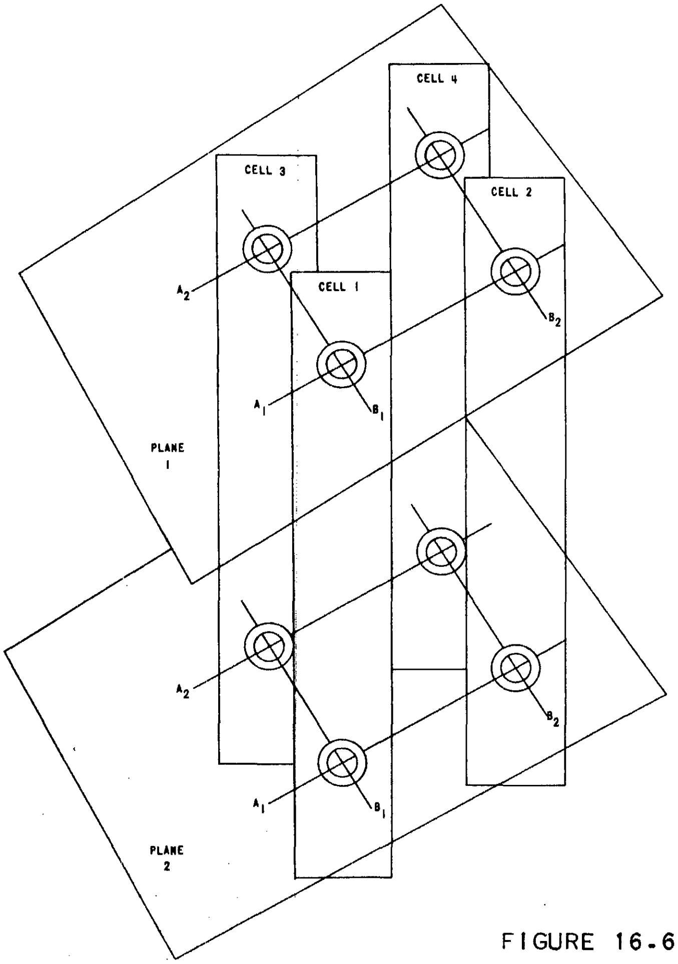


FIGURE 16-6

The first bit positions of the words are stored in plane 1, the second bit positions, in plane 2. To store or read information in or out of cell 1, a half jolt of current is sent along lines A1 and B1 in both planes. Similarly, lines A1 and B2 are associated with cell 2; lines A2 and B1, with cell 3; and A2 and B2, with cell 4.

Univac's memory works in an analogous way, except that the memory consists of 84 planes each containing 2000 cores. In Univac the diameter of a core is about .06 inch. Besides the memory, rW, rZ, rI and rO consist of cores, thus providing compact storage that permits rapid access.

SUMMARY

The storage medium of rA, rX, rL, CC and CR is mercury tank. The storage medium of the memory, rW, rZ, rI and rO is core.

chapter 17

Manipulation of Information

REPRESENTATION OF INFORMATION

Any positive number can be represented by a row of marks such as

111111111 (or 9)

although all but the smallest numbers become unwieldy in such notation. For ease of manipulation a positional notation using symbols to represent different rows of marks is more convenient. One such notation is the Arabic, which uses ten different symbols or digits, 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.

The number of different digits used in a positional notation or system is known as the base of the system. Thus, Arabic notation is known as a base ten or decimal system. Using one digit position, quantities as large as nine can be represented in the decimal system. To represent a quantity larger than nine another digit position must be used. Thus, to represent the quantity ten a carry is made into the digit position to the left and the original digit position reverts to zero. The expansion of this system is exemplified by the speedometer of a car.

In positional notation each digit position, or column, implies a power of the base as a multiplier of the digit in the column. The decimal number 1076 is positional notation for the expression

$$(1 \text{ x } 1000) + (0 \text{ x } 100) + (7 \text{ x } 10) + (6 \text{ x } 1)$$

The columns imply powers of ten,

$$\begin{array}{rcll}
1 & = & 1 & = 10^{o} \\
10 & = & 10 & = 10^{1} \\
100 & = & 10 \text{ x } 10 & = 10^{2} \\
1000 & = & 10 \text{ x } 10 \text{ x } 10 & = 10^{3}
\end{array}$$

and, appropriately enough, are named the units column, the tens column, the hundreds column, and so on.

A computer that represents numbers in decimal notation must have storage elements capable of assuming ten easily distinguishable, stable states, one for each possible digit. While such elements exist, their cost prohibits the construction of a computer that represents numbers in decimal notation. Electronic elements lend themselves most naturally to two stable state devices. Thus, computers usually represent numbers in the base two or binary system. The binary system can be built up in a way analogous to the decimal. There are two possible digits, 0 and 1, used in conjunction with successive powers of two

$$\begin{array}{rcl}
2^{0} & = & 1 \\
2^{1} & = & 2 \\
2^{2} & = & 4 \\
2^{3} & = & 8 \\
. & & . \\
. & & . \\
. & & .
\end{array}$$

Thus, the binary equivalent of a decimal nine is 1001, which is binary notation for the expression

$$(1 \text{ x } 8) + (0 \text{ x } 4) + (0 \text{ x } 2) + (1 \text{ x } 1)$$

STUDENT EXERCISES

Write the binary equivalents of the decimal numbers 6, 13, 15, 27 and 43.

BINARY ADDITION

The addition table for the binary system is

$$\begin{aligned} 0 + 0 &= 0 \\ 0 + 1 &= 1 \\ 1 + 1 &= 10 \\ 1 + 1 + 1 &= 11 \end{aligned}$$

The sum of two binary ones is the binary number 10, the binary equivalent of a decimal two. The binary number 10 is not what is called ten, which is a decimal, not a binary, number. Similar remarks hold for the sum of three binary ones, which is the binary number 11, not the decimal number eleven.

Example

DECIMAL	BINARY
13	^{1}1101
14	1110
27	11011

STUDENT EXERCISES

Add the following

1011	1010	11001
1111	10111	10111

ADDITION OF TWO NUMBERS WITH OPPOSITE SIGNS

While addition of two numbers with opposite signs could be done by use of a "subtraction" table, computers use the method of complementation. For any given number there exists a second number which when added to the first will produce a sum consisting of a one followed by as many zeros as there are digits in the first number. The second number is the complement of the first. To get the complement of a binary number

1. Replace the ones with zeros and the zeros with ones and
2. Add a binary one to the result.

For example, given

$$1101$$

replace ones with zeros and zeros with ones, and add a binary one

$$\begin{array}{lr} & 0010 \\ & \underline{1} \\ \text{Complement} & 0011 \end{array}$$

Proof: $1101 + 0011 = 10000$

To add two numbers with opposite signs

1. consider as many digit positions of the smaller in absolute value of the two numbers as there are in the larger by inserting as many nonsignificant zeros as are necessary
2. take the complement of the absolute value of the smaller in absolute value
3. Add the absolute value of the other number to the result
4. drop the most significant carry

and 5. prefix the sign of the larger in absolute value.

Example

$$\begin{array}{lr} \text{ADD} & -101101 \\ & \underline{+1011} \end{array}$$

$$\begin{array}{lr} \text{STEP 1.} & -101101 \\ & \underline{+001011} \end{array}$$

STEP 2. The smaller in absolute value is

$$001011$$

$$\begin{array}{r} 110100 \\ \underline{1} \\ 110101 \end{array}$$

$$\begin{array}{lr} \text{STEP 3.} & 101101 \\ & \underline{110101} \\ & 1100010 \end{array}$$

STEP 4. 100010

STEP 5. - 100010

Translating the numbers into decimal notation, the steps form the expression

$$\underbrace{-}_{\text{Step 5}}\ (\underbrace{64 - 11}_{\text{Steps 1 and 2}}\ + \underbrace{45}_{\text{Step 3}} - \underbrace{64}_{\text{Step 4}})$$

The expression reduces to - 45 + 11, which is the decimal equivalent of the original problem.

STUDENT EXERCISES

Add the following

$$\begin{array}{r} -1011 \\ +1111 \\ \hline \end{array} \qquad \begin{array}{r} +\ 1010 \\ -\ 10111 \\ \hline \end{array} \qquad \begin{array}{r} -11001 \\ -10111 \\ \hline \end{array}$$

SUBTRACTION

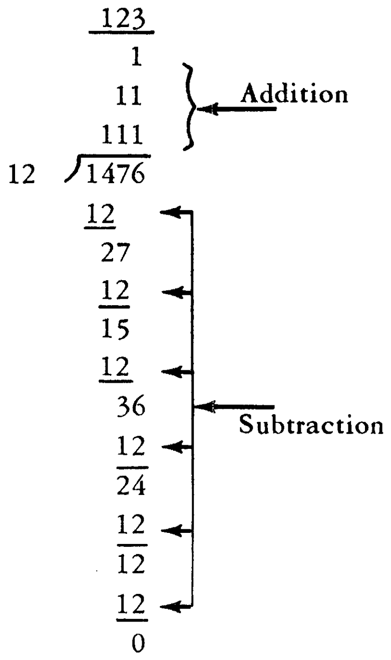
Subtraction is to change the sign of the subtrahend and add the result to the minuend. Thus, subtraction can be done by addition.

MULTIPLICATION

Binary multiplication can be done by a series of additions just as decimal multiplication can, as shown on pages 153 . Thus, multiplication can be done by addition.

DIVISION

Binary division can be done by a series of additions and subtractions just as decimal division can.



Thus, division can be done by addition.

THE ARITHMETIC UNIT

The above discussion demonstrates that the arithmetic unit of a binary computer can be made if

1. a binary adder,
2. a complementer,
3. a sign changer,
4. a counter to keep track of the additions and subtractions in multiplication and division.
5. an equality comparator,

and 6. a magnitude comparator
can be made.

CODED BINARY

Binary representation is used in computers in one of two forms. The first is the binary notation just described called pure binary representation. The other is called coded binary representation. In this representation, only the pure binary equivalents of the ten decimal digits are used.

DECIMAL	PURE BINARY
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

Any decimal number greater than nine is represented by a combination of the above codes. For example, the decimal number 147 would be represented as

0001 0100 0111

If a computer uses pure binary representation either

1. input must be prepared and output received in pure binary representation

or 2. the computer must be able to convert input received in decimal representation to pure binary representation and convert output produced in pure binary to decimal.

The first approach is awkward and the second requires expensive hardware. But if a computer uses coded binary representation, data may be introduced and removed from the computer directly in decimal representation.

The Central Computer of the Univac Data-Automation System uses a modification of coded binary representation called excess three representation. In this representation each decimal digit is represented by the binary equivalent of the digit plus a decimal three.

DECIMAL	EXCESS THREE (XS-3)
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

Excess three representation has three advantages over straight coded binary.

1. The addition of two excess three numbers produces a carry if and only if the addition of their decimal equivalents produces a carry.
2. An excess three number can be complemented in the same way a pure binary number is.
3. Excess three representation provides three digits,0000 ,0001 and 0010, whose decimal equivalents have a value less than zero.

EXCESS THREE ARITHMETIC

If two like signed excess three digits are added, the decimal equivalent of their sum is not equal to the sum of the decimal equivalents of the digits.

DECIMAL	EXCESS THREE
5	1000
1	0100
6	1100

The decimal equivalent of the excess three digit 1100 is not six, but nine. The reason for this fact is that if the addition does not produce a carry the sum is not in excess three representation, but in excess six notation. To convert the sum to excess three representation it is necessary to subtract the pure binary equivalent of a decimal three, 0011. The complement of the pure binary number 0011 is 1101. Thus, to "correct" the sum of two excess three digits that do not produce a carry, add the pure binary number 1101 to the sum.

$$\begin{array}{r} 1100 \\ \underline{1101} \\ 1001 \end{array}$$

The excess three digit 1001 is the equivalent of a decimal six.

However, if the addition of two excess three digits produces a carry, the sum is not represented in excess six.

$$\begin{array}{rcr} 5 & \qquad\qquad & 1000 \\ \underline{6} & & \underline{1001} \\ \hookleftarrow 1 & & \hookleftarrow 0001 \end{array}$$

The arrow indicates a carry.

The reason for the above fact is that the carry in the excess three addition carries the equivalent of a decimal 16 out of the sum. Ten of this 16 is the decimal carry to the next column, which is desired; three of the 16 is what previously produced an excess six sum, and its carry is of no concern; but the last three is what was necessary to produce an excess three sum. Thus, the sum comes out in pure binary representation. To convert the sum to excess three representation it is necessary to add the pure binary equivalent of a decimal three, 0011, to the sum.

$$\begin{array}{r} 0001 \\ \underline{0011} \\ 0100 \end{array}$$

The excess three digit 0100 is the equivalent of a decimal one.

In summary, to add two like signed excess three numbers

1. add the numbers according to the rules of pure binary addition

and 2. apply "correction factors" to each digit in the sum. The correction factors are as follows:

   a. If the column in which the digit appears did not produce a carry, add the correction factor 1101.

   b. If the column produced a carry, add 0011.

For example,

$$\begin{array}{lccc}
 & 0100 & 0111 & 1010 \\
 & 1001 & 0110 & 1000 \\
\hline
\text{Intermediate sum -} & 1101 & 1110 \curvearrowleft & 0010 \\
\text{Correction factors -} & 1101 & 1101 & 0011 \\
\hline
\text{Excess three sum -} & 1010 & 1011 & 0101
\end{array}$$

Since the correction factors apply only to individual digits and not the entire sum, any carry produced is ignored.

STUDENT EXERCISES

Add

$$\begin{array}{ccc}
0110 & 1010 & 0011 \\
0111 & 1000 & 1010 \\
\hline
\end{array}$$

$$\begin{array}{ccc}
1000 & 0101 & 1100 \\
1010 & 0111 & 0101 \\
\hline
\end{array}$$

ADDITION OF TWO EXCESS THREE NUMBERS WITH OPPOSITE SIGNS

Two excess three numbers with opposite signs are added in the same way as two pure binary numbers with opposite signs are.

Example

$$\begin{array}{lcccc}
\text{ADD} & -1010 & 0111 & 1001 & 0100 \\
 & + & 0100 & 0110 & 1100 \\
\hline
\text{STEP 1.} & -1010 & 0111 & 1001 & 0100 \\
 & +0011 & 0100 & 0110 & 1100 \\
\hline
\text{STEP 2.} & 1100 & 1011 & 1001 & 0011 \\
 & & & & 1 \\
\hline
 & 1100 & 1011 & 1001 & 0100
\end{array}$$

STEP 3.

$$\begin{array}{rrrr} 1010 & 0111 & 1001 & 0100 \\ 1100 & 1011 & 1001 & 0100 \\ \hline \nwarrow 0111 & \nwarrow 0011 & \nwarrow 0010 & 1000 \\ 0011 & 0011 & 0011 & 1101 \\ \hline \nwarrow 1010 & 0110 & 0101 & 0101 \end{array}$$

STEP 4.

$$\begin{array}{rrrr} 1010 & 0110 & 0101 & 0101 \end{array}$$

STEP 5.

$$\begin{array}{rrrr} -1010 & 0110 & 0101 & 0101 \end{array}$$

STUDENT EXERCISES

ADD

$$\begin{array}{rrr} +1010 & 0011 & 0111 \\ -0101 & 0011 & 1010 \\ \hline \end{array}$$

$$\begin{array}{rrr} -0111 & 1011 & 1100 \\ +1000 & 0110 & 0101 \\ \hline \end{array}$$

ALPHABETIC REPRESENTATION

To represent the ten decimal digits and the 26 alphabetics, the computer must have 36 different representations. Four binary digit (bit) positions only allow 16 different representations. Consequently, to the excess three part of the representation of a character a two bit position zone is added, allowing a maximum of 64 different representations as shown in figure 9-2.

LOGICAL BUILDING BLOCKS

For purposes of representation, certain electronic circuits can be represented by symbols. A wire is shown as an arrow. It is assumed that current can flow along the wire only in the direction indicated by the arrowhead.

THE FLIPFLOP

A flipflop (FF) is shown as a rectangle having two input lines and two outputs.

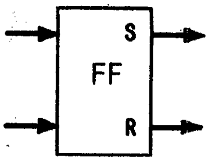


FIGURE 17-1

One input and one output are associated with one side of a FF, called the set side (S); the other input and output, with the other side, called the reset side (R). A FF has the following characteristics. It always emits a steady voltage, called a signal, the signal coming from the set side or the reset side, but never from both. If the signal is coming from the set side, the FF is said to be set; from the reset side, reset. If a FF is reset, a pulse applied to the set side will set the FF. The FF will then remain set until a pulse is applied to the reset side, which will reset the FF, and vice versa.

THE GATE

A gate (G) is shown as a square having one output and one or more inputs.

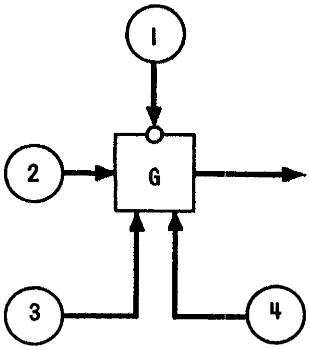


FIGURE 17-2

The inputs are of two types, permissive inputs, shown as arrows with the arrowheads touching the G; and inhibitory, shown as arrows with circles between the G and the arrowheads.

Some inputs are identified by circles with numbers in them, the circle representing the source of the input.

A G emits a signal on the output under the influence of signals on the inputs. When a G emits a signal the G is said to be permissed. A signal on an inhibitory input or the absence of a signal on a permissive input prevents a G from being permissed. Thus, a G is permissed only if all permissive signals are present; and all inhibitory signals, absent. The G in Figure 17-2 is permissed only when signals 2, 3 and 4 are present; and signal 1, absent. The G in the following figure is permissed when signal 1 and signal 2 or 3 are present.

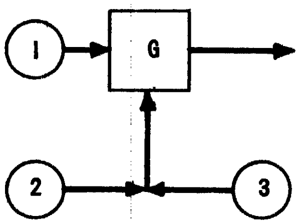


FIGURE 17.3

THE BINARY COUNTER

A binary counter (BC) is shown as a rectangle having three inputs and three outputs.

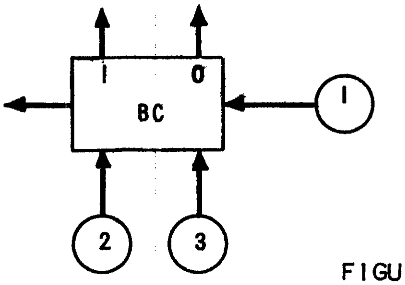


FIGURE 17.4

One input and one output are associated with one side of a BC, called the zero side; another input and output, with the other side, called the one side. A BC always emits a signal from either the zero or one side. If the signal is coming from the zero side, the BC is said to be in the zero state; from the one side, in the one state. A pulse applied to the zero side will put the BC in the zero state; applied to the one side, in the one state.

The third input to a BC is called the stepping input. If a BC is in the zero state, a pulse on the stepping input will put the BC in the one state, and vice versa. Each time a BC changes state it is said to be stepped. Everytime a BC is stepped from the one to the zero state, it emits a pulse on the third output, called the carry output.

One use of a BC is as an odd even checker.

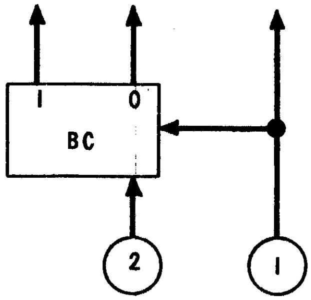


FIGURE 17.5

If a pulse from source 2 initially puts the BC in the zero state, and a character is then transferred from source 1, at the end of the transfer the BC will be in the one state if the character contained an odd number of ones, in the zero state if it contained an even number.

BC's can also be connected together.

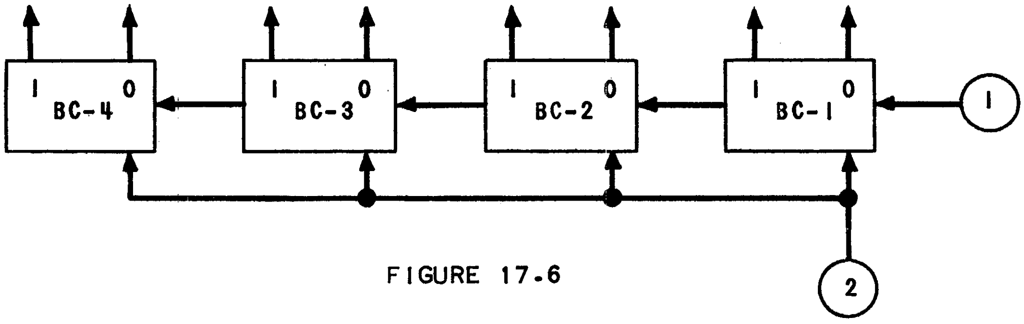


FIGURE 17-6

The above string of BC's will count in pure binary from 0000 to 1111.

THE DELAY

A delay is shown as a rectangle containing a D with a numeric subscript.

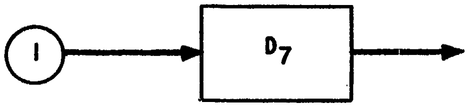


FIGURE 17-7

The subscript indicates the number of pulse times needed for a pulse on the input to emerge on the output, where a pulse time is the time needed for a pulse to pass a given point in the circuitry. A mercury tank is a type of delay.

LOGICAL CIRCUITS

The following circuits are built to operate on unsigned pure binary numbers in the form of a pulse combination traveling least significant bit first.

MAGNITUDE COMPARATOR

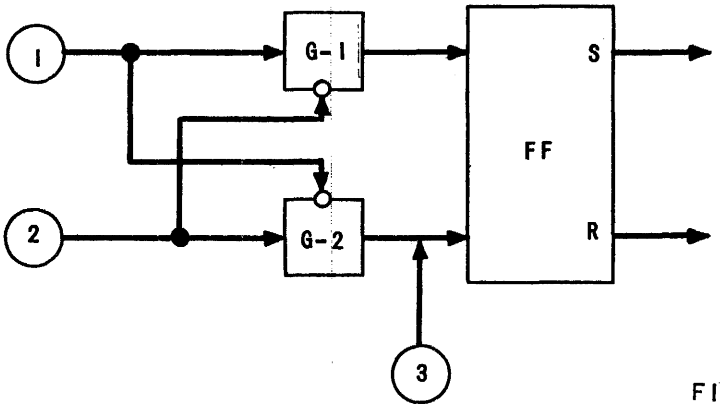


FIGURE 17-8

If a pulse from source 3 initially resets the FF, and a number A is then transferred from source 1 at the same time as another number B is transferred from source 2, at the end of the transfer the FF will be set if A is greater than B and reset if A is less than or equal to B. When A and B have an identical column neither G-1 nor G-2 is permissed, and the FF remains in its present state. When A has a one in a column in which B has a zero, G-2 is inhibited, but G-1 is permissed, and the FF is set. In the reverse situation, G-1 is inhibited, but G-2 is permissed, and the FF is reset.

HALF ADDER

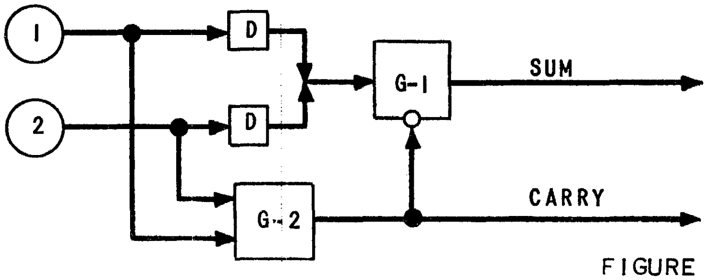


FIGURE 17-9

If a number A is transferred from source 1 at the same time as another number B is transferred from source 2, the sum without carry of A and B will be produced on the sum output, and the carry will be produced on the other output, called the carry output.

A	0011
B	0101
Sum without carry	0110
Carry	0001

If both A and B have a zero in the same column, neither G-1 nor G-2 is permissed, and a zero is produced on both the sum and carry outputs. If either A or B has a one in a column in which the other has a zero, G-1 is permissed, and a one is produced on the sum output, but G-2 is not permissed, and a zero is produced on the carry output. If both A and B have a one in the same column, G-2 is permissed. (The delays are minute but insure G-2 output before G-1 output.) G-2 output is a binary 1, a carry, which inhibits G-1, so that the sum of the bits is binary zero.

A half adder (HA) is customarily represented as in the figure below.

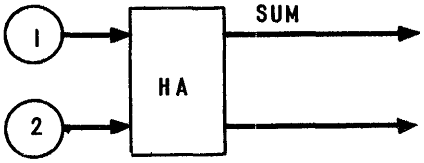


FIGURE 17-10

If a number consisting of all ones is transferred from source 2 at the same time as a number A is transferred from source 1, the HA will act as a complementer, producing a number on the sum output that has ones where A has zeros and zeros where A has ones.

A HA is used as the "sign changer" for the subtract and negative multiplication instructions. The binary configuration 1000001 is added to the characters in the sign position of the word transferred to rX and the sum, without carry, becomes the new sign. Thus:

1000001	1000001	1000001	1000001
(-) 0000010	(+) 1000011	(A) 1010100	(B) 0010101
(+) 1000011	(-) 0000010	(B) 0010101	(A) 1010100

The HA is also used in the circuitry of

1. a binary adder

and 2. an equality comparator.

BINARY ADDER

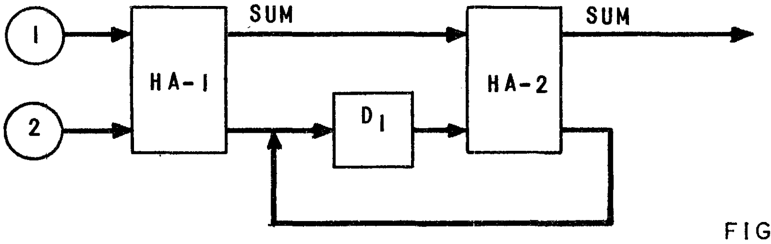


FIGURE 17-11

HA-1 produces the sum without carry on its sum output, and HA-2 adds in the carry to produce a sum with carry on its sum output. The delay of one assures that the carry is added in the right digit positions.

EQUALITY COMPARATOR

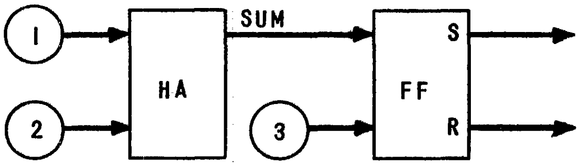


FIGURE 17-12

If a pulse from source 3 initially resets the FF, and a number A is then transferred from source 1 at the same time as another number B is transferred from source 2, at the end of the transfer the FF will be reset if A equals B and set if A is not equal to B. As long as A and B have identical columns no pulse is produced on the sum line, and the FF remains in the reset state. However, as soon as either A or B has a one in a column in which the other has a zero a pulse is produced on the sum line, and the FF is set. Once set the FF remains set for the rest of the comparison.

chapter 18

Insuring Accuracy of Processing

In any data processing system one of the chief concerns is the accuracy of the results. In a computer data processing system, errors may be introduced in one of three ways.

1. Erroneous data fed into the system.
2. Erroneous intervention by an operator into the system.
3. Malfunctioning of the computer.

INPUT DATA ACCURACY

The accuracy of input data can be assured by validity runs, described on pages 256 and 257.

OPERATOR ACCURACY

There are points at which an operator must manually intervene in the otherwise automatic operation of a computer. For example, to run a routine, an operator must mount input tapes. The stored program allows the computer to check all operator interventions for accuracy. For example, by convention, the first block of each input tape contains, not data, but an identification of the data on the tape. By means of this identification block, the computer can check that the data mounted is actually the data associated with the stored routine.

RERUN

Rerun is designed to handle situations where processing is interrupted during a run. Power failure or removal of a routine for one of higher priority are examples of such interruptions. Rerun consists of periodically writing, or dumping, the contents of the memory on tape. Then, no matter where processing is interrupted, it can be restarted at the point of the last memory dump by using the memory dump to reconstitute the memory. Rerun eliminates the necessity to restart an interrupted run from the beginning, thus conserving computer time.

COMPUTER ACCURACY

In computers every pulse has a significance which, if lost, alters the content of the whole message. A power failure of only .4 μs duration can cause the loss of a binary one. Such a loss could change a six to a five.

DECIMAL	EXCESS THREE WITH ZONE
6	001001
5	001000

If such a situation occurred when two words were being compared, the comparator may indicate inequality when equality is the case.

If such an error occurred when the key 60032 is being checked for equality between files A and B in figure 18-1, no item following the item with key 60029 would be processed, since the computer would exhaust file B in a vain search for equality of keys.

FILE A	FILE B
50031	50031
50032	50032
59999	59999
60028	60028
60029	60029
60032 → 50032	60032
60034	60034
.	.
.	.
.	.

FIGURE 18-1

No malfunction can be tolerated in a computer, since even a minute failure may have disastrous results.

TYPES OF FAILURES

Errors can be produced by permanent or intermittent failures of equipment. A blown fuse is an example of a permanent failure. A gradually weakening tube that sometimes overloads under the influence of a particular pulse combination is an example of an intermittent failure.

ERROR DETECTION

It is not possible to build a computer that will never malfunction. The only solution is to provide some means of detecting errors as they occur and preventing the the propagation of the error. The responsibility for detecting errors can be placed on the programmer or checks can be built into the computer.

PROGRAMMED ERROR DETECTION

DIAGNOSTIC ROUTINES

A computer can execute a routine the output of which is known. If the output is as expected, the routine guarantees that the computer has not developed a permanent failure. However, the routine provides no assurance that an intermittent failure will not occur during a production run. Moreover, running time for the routine is lost time as far as production is concerned.

DUPLICATE RUNS

After a computer has executed a production routine, it can execute the routine a second time. The results of the runs can be compared, the computer usually being used to make the comparison. If the comparison checks out, and if a permanent failure has not developed since the last diagnostic run, the output is correct, Such an approach more than doubles, and may more than triple, the computer time required to produce the output. Moreover, if the comparison does not check out, it is impossible to know if a failure occurred during the first or second production run or during the comparison or during any combination of the three.

PROGRAMMED CHECKS

The production routine can be programmed in such a manner that, immediately after the execution of a subroutine, a second subroutine, checking the results of the first for accuracy, is executed. For example,

```
0010  B01880
               A-1881   }   addition
0011  H01882
               S-1880
0012  L01881            }   check
               Q00020
```

If control is transferred to cell 0020, the addition was correct; if control passes to cell 0013, incorrect.

Programmed checks increase the running time of a production routine by a factor of at least two thirds. The increase in memory space required by the programmed checks is even more drastic. Moreover, there are operations that do not lend themselves to a programmed check. Selection of the next instruction to be executed and selection of the cell specified by an instruction are examples of such operations. By themselves, programmed checks cannot assure output accuracy.

If a computer failure occurs, the failure must be corrected before the computer can return to operation. Thus, the fault must be located in the computer hardware. Since programmed error detection may not stop the computer at the point when an error occurs, this method provides little or no help to the technician in locating the fault. The time required for the technician to locate the fault further reduces productive computer time.

BUILT IN CHECKS

Checking circuits can be built into a computer in such a manner that the computer stops the instant an error occurs and lights a neon on the control panel, thus indicating the nature of the error. These circuits operate in conjunction with the processing circuits. No computer time is lost because of the existence of checking circuits. Admittedly, checking circuits cost money, but they save

1. productive computer time lost because of diagnostic runs,
2. productive computer time lost because of duplicate operation, either by duplicate runs or by programmed checks,
3. productive computer time lost because runs must be subdivided to provide memory space for programmed checks,
4. productive computer time lost because the computer does not stop the instant the error occurs, thus requiring the technician to locate the fault with little or no help from the checking routines.
5. productive computer time lost because of errors that escape programmed checks,
6. company embarrassment caused by such errors

and 7. productive programmer time lost in the search for the elusive perfect program check.

Built in checks represent a fixed initial cost; checking routines, a continual, and basically, hidden cost. It is estimated that built in checks will pay for themselves in less than a year.

BUILT IN CHECKS OF THE UNIVAC CENTRAL COMPUTER

ODD EVEN CHECK

The odd even checker is a reliable, inexpensive checking circuit which checks against the proper storage of data and the proper transfer of data from one storage to another. There is an odd even check located

1. on the High-Speed Bus (HSB) which is the transmission line between the registers and the memory,
2. on each of the adder inputs,
3. between the Uniservos and rI

and 4. between rO and the Uniservos.

However, there are failures that the odd even check cannot detect. For this reason duplicate, counting and logical checks are also used.

DUPLICATED CIRCUITRY

Several elements of the Central Computer of the Univac System are duplicated. In the case of storage or transmission elements, such as the registers and the HSB, the contents of the duplicated elements are continuously compared for identity. In the case of processing elements, such as the adder and comparator, equality of output is the basis of the check. The duplicated elements are

1. the HSB,
2. each of the adder inputs,
3. the adder,
4. rA,
5. rL,
6. rX,
7. rF,
8. the comparator,
9. the memory counter, in which the address in SR is stored, and which advances the cell addresses during the execution of multiword transfer and tape instructions,

and 10. the Time Out circuits, which determine whether Univac is on TO or Time On.

COUNTING CHECKS

The representation

1111111

contains an odd number of ones, but for electronic reasons, is not used. To prevent such a representation from passing across the HSB, an all ones detector is located on the HSB.

A counter checks that no more nor less than 720 characters are transferred to rI during the execution of a read instruction.

LOGICAL CHECKS

When storing a one in a core, during phase II the core flips from the zero to one state, thus producing current on the output line. This current is used to detect the incorrect storage of a one in a core.

When transferring a word from the arithmetic or control units to the memory, a pulse, called a staticizing pulse, is required to gate the word from the HSB to the memory. The production of the staticizing pulse is checked.

Checks are also made to determine that

1. the proper memory cell is selected at the beginning of the execution of an instruction,
2. only one cell is selected during the execution of instructions involving the memory,
3. an instruction is legitimate,
4. the address specified by an instruction is legitimate,
5. an instruction is properly executed,
6. the proper cell in rW or rZ is selected at the beginning of the execution of a multiword transfer instruction,
7. one cell in rW or rZ is selected during each stage of the execution of a multiword transfer instruction

and 8. the cycling unit, which synchronizes the computer's operation, is functioning properly.

INPUT - OUTPUT CHECKS

(The checks employed in the input-output units have been described in Chapter XIII.)

A counter is used to control the transfer of data from rO to the Uniservos and from the Uniservos to rI.

A check is made to determine that the counter is properly set at the beginning of the execution of a tape instruction.

The counter produces signals to control the transfer of the data. A check is made against the proper production of the signals.

Additional checks are made to determine that

1. the proper cell in rO is selected at the beginning of the execution of a write instruction,
2. one cell in rO is selected during each word transfer in the execution of a write instruction,

3. the proper cell in rI is selected at the beginning of the execution of a read instruction,
4. one cell in rI is selected during each word transfer in the execution of a read instruction,
5. a sprocket pulse is read every time a character is read

and 6. only one Uniservo is selected during the execution of a tape instruction.

The interlock circuitry produces signals that inhibit

1. the initiation of a read instruction while rI is engaged in the transfer of data.
2. the initiation of a write instruction while rO is engaged in the transfer of data.

and 3. the initiation of tape instruction while the tape specified is moving or rewound with interlock.

A check is made against the proper functioning of the interlock circuitry.

THE EFFECT OF ERRORS

If an error is detected in any part of the computer other than the input-output circuitry, the computer immediately stalls. If an error is detected in the input-output circuitry, the computer stalls as soon as another attempt to use the faulty part of the circuitry is made. For example, if an error was detected during the writing of a block on T6, the computer would stop as soon as another write instruction or another tape instruction involving T6 was transferred to SR. Given an error, such a situation prevents the computer from propagating the error. In either case, as soon as an error occurs, a neon on the Supervisory Control Panel lights, indicating the specific error that has occurred.

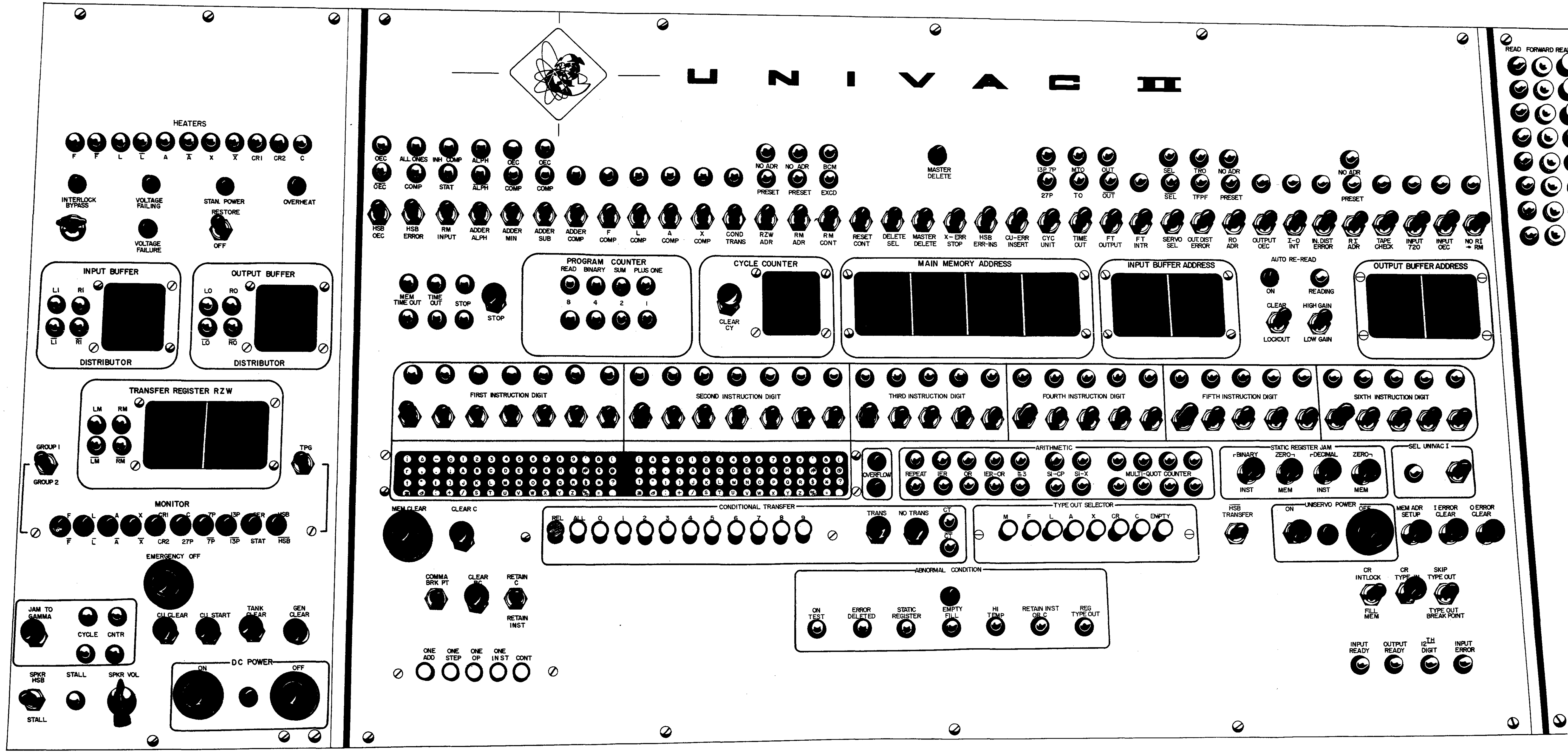
BUILT IN CHECKS ON THE PROGRAM

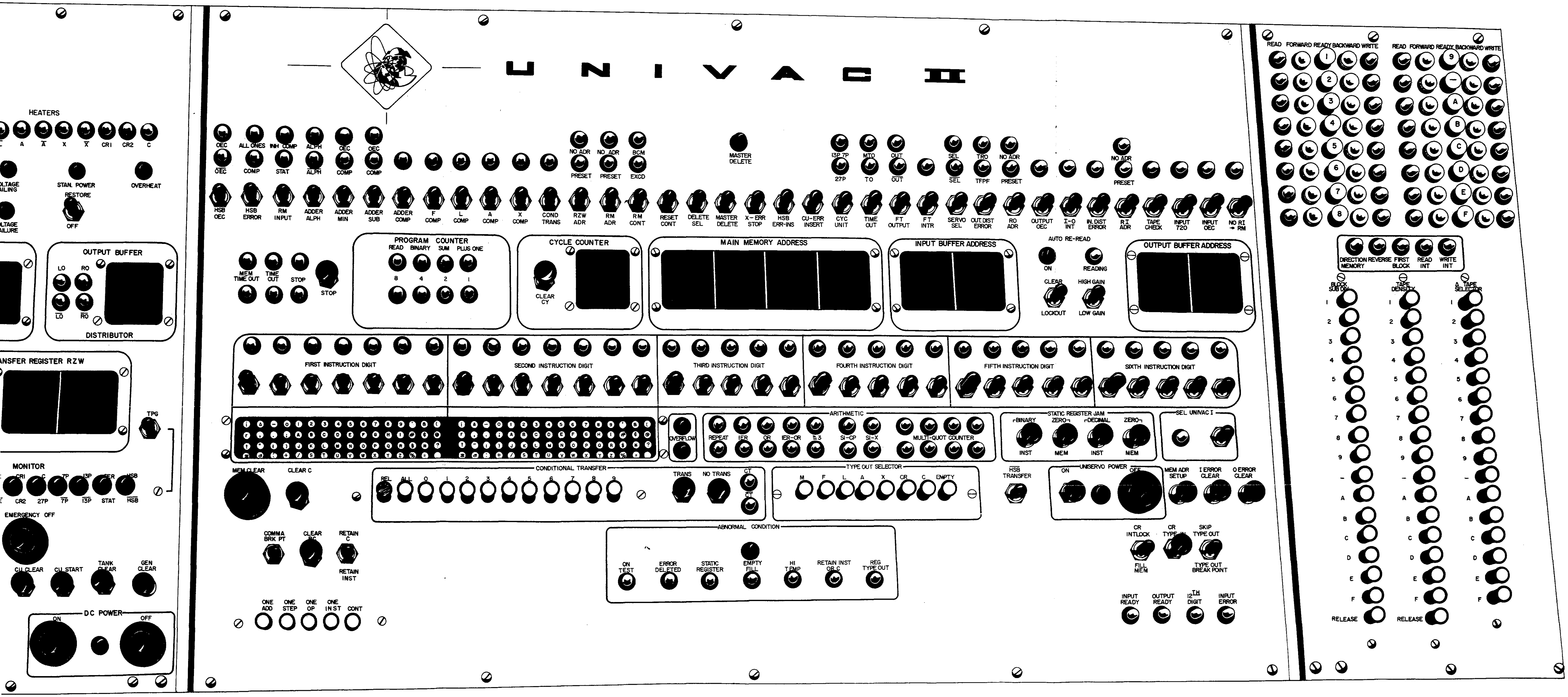
Besides checking the accuracy of its operations, the computer also checks for the occurrence of

1. an adder-alph error,

and 2. the attempt, after reading a given block into rI, to read another block into rI without first transferring the given block from rI to the memory.

In either case, the computer immediately stalls and lights an appropriate neon on the Supervisory Control Panel.


UNIVAC II
HEATERS
F
$\overline{F}$
L
$\overline{L}$
A
$\overline{A}$
X
$\overline{X}$
CR1
CR2
C
INTERLOCK BYPASS
VOLTAGE FAILING
STAN. POWER
OVERHEAT
RESTORE
VOLTAGE FAILURE
OFF
INPUT BUFFER
LI
RI
$\overline{LI}$
$\overline{RI}$
DISTRIBUTOR
OUTPUT BUFFER
LO
RO
$\overline{LO}$
$\overline{RO}$
DISTRIBUTOR
TRANSFER REGISTER RZW
LM
RM
$\overline{LM}$
$\overline{RM}$
GROUP 1
GROUP 2
TPG
MONITOR
F
L
A
X
CR1
C
7P
13P
SER
HSB
$\overline{F}$
$\overline{L}$
$\overline{A}$
$\overline{X}$
CR2
27P
$\overline{7P}$
$\overline{13P}$
STAT
$\overline{HSB}$
EMERGENCY OFF
JAM TO GAMMA
CYCLE
CNTR
CU CLEAR
CU START
TANK CLEAR
GEN CLEAR
DC POWER
ON
OFF
SPKR HSB
STALL
STALL
SPKR VOL
OEC
ALL ONES
INH COMP
ALPH
OEC
OEC
$\overline{OEC}$
COMP
STAT
$\overline{ALPH}$
COMP
COMP
NO ADR
NO ADR
BCM
PRESET
PRESET
EXCD
MASTER DELETE
13P 7P
MTO
OUT
27P
TO
$\overline{OUT}$
SEL
TRO
NO ADR
$\overline{SEL}$
TFPF
PRESET
NO ADR
PRESET
HSB OEC
HSB ERROR
RM INPUT
ADDER ALPH
ADDER MIN
ADDER SUB
ADDER COMP
F COMP
L COMP
A COMP
X COMP
COND TRANS
RZW ADR
RM ADR
RM CONT
RESET CONT
DELETE SEL
MASTER DELETE
X-ERR STOP
HSB ERR-INS
CU-ERR INSERT
CYC UNIT
TIME OUT
FT OUTPUT
FT INTR
SERVO SEL
OUT. DIST ERROR
RO ADR
OUTPUT OEC
I-O INT
IN. DIST ERROR
RI ADR
TAPE CHECK
INPUT 720
INPUT OEC
NO RI → RM
MEM TIME OUT
TIME OUT
STOP
STOP
PROGRAM COUNTER
READ
BINARY
SUM
PLUS ONE
8
4
2
1
CYCLE COUNTER
CLEAR CY
MAIN MEMORY ADDRESS
INPUT BUFFER ADDRESS
AUTO RE-READ
ON
READING
CLEAR
HIGH GAIN
LOCKOUT
LOW GAIN
OUTPUT BUFFER ADDRESS
FIRST INSTRUCTION DIGIT
SECOND INSTRUCTION DIGIT
THIRD INSTRUCTION DIGIT
FOURTH INSTRUCTION DIGIT
FIFTH INSTRUCTION DIGIT
SIXTH INSTRUCTION DIGIT
OVERFLOW
ARITHMETIC
REPEAT
IER
OR
IER-OR
≧3
SI-CP
SI-X
MULTI-QUOT COUNTER
STATIC REGISTER JAM
BINARY
ZERO
DECIMAL
ZERO
INST
MEM
INST
MEM
SEL UNIVAC I
MEM CLEAR
CLEAR C
CONDITIONAL TRANSFER
REL
ALL
0
1
2
3
4
5
6
7
8
9
TRANS
NO TRANS
CT
$\overline{CT}$
TYPE OUT SELECTOR
M
F
L
A
X
CR
C
EMPTY
$\overline{HSB}$ TRANSFER
UNISERVO POWER
ON
OFF
MEM ADR SETUP
I ERROR CLEAR
O ERROR CLEAR
COMMA BRK PT
CLEAR RC
RETAIN C
RETAIN INST
ABNORMAL CONDITION
ON TEST
ERROR DELETED
STATIC REGISTER
EMPTY FILL
HI TEMP
RETAIN INST OR C
REG TYPE OUT
CR INTLOCK
CR TYPE IN
SKIP TYPE OUT
FILL MEM
TYPE OUT BREAK POINT
ONE ADD
ONE STEP
ONE OP
ONE INST
CONT
INPUT READY
OUTPUT READY
12TH DIGIT
INPUT ERROR
READ
FORWARD


UNIVAC II
HEATERS
A
A
X
X
CR1
CR2
C
VOLTAGE FAILING
STAN. POWER
OVERHEAT
RESTORE
OFF
VOLTAGE FAILURE
OUTPUT BUFFER
LO
RO
LO
RO
DISTRIBUTOR
TRANSFER REGISTER RZW
TPG
MONITOR
CR1
C
7P
13P
SER
HSB
CR2
27P
7P
13P
STAT
HSB
EMERGENCY OFF
CU CLEAR
CU START
TANK CLEAR
GEN CLEAR
DC POWER
ON
OFF
OEC
OEC
ALL ONES
COMP
INH COMP
STAT
ALPH
ALPH
OEC
COMP
OEC
COMP
NO ADR
PRESET
NO ADR
PRESET
BCM
EXCD
MASTER DELETE
13P 7P
27P
MTO
TO
OUT
OUT
SEL
SEL
TRO
TFPF
NO ADR
PRESET
NO ADR
PRESET
HSB OEC
HSB ERROR
RM INPUT
ADDER ALPH
ADDER MIN
ADDER SUB
ADDER COMP
F COMP
L COMP
A COMP
X COMP
COND TRANS
RZW ADR
RM ADR
RM CONT
RESET CONT
DELETE SEL
MASTER DELETE
X-ERR STOP
HSB ERR-INS
CU-ERR INSERT
CYC UNIT
TIME OUT
FT OUTPUT
FT INTR
SERVO SEL
OUT. DIST ERROR
RO ADR
OUTPUT OEC
I-O INT
IN. DIST ERROR
RI ADR
TAPE CHECK
INPUT 720
INPUT OEC
NO RI → RM
MEM TIME OUT
TIME OUT
STOP
STOP
PROGRAM COUNTER
READ
BINARY
SUM
PLUS ONE
8
4
2
1
CYCLE COUNTER
CLEAR CY
MAIN MEMORY ADDRESS
INPUT BUFFER ADDRESS
AUTO RE-READ
ON
READING
CLEAR
LOCKOUT
HIGH GAIN
LOW GAIN
OUTPUT BUFFER ADDRESS
FIRST INSTRUCTION DIGIT
SECOND INSTRUCTION DIGIT
THIRD INSTRUCTION DIGIT
FOURTH INSTRUCTION DIGIT
FIFTH INSTRUCTION DIGIT
SIXTH INSTRUCTION DIGIT
OVERFLOW
ARITHMETIC
REPEAT
IER
OR
IER-OR
≧3
SI-CP
SI-X
MULTI-QUOT COUNTER
STATIC REGISTER JAM
BINARY
ZERO
DECIMAL
ZERO
INST
MEM
INST
MEM
SEL UNIVAC I
MEM CLEAR
CLEAR C
CONDITIONAL TRANSFER
REL
ALL
0
1
2
3
4
5
6
7
8
9
TRANS
NO TRANS
CT
CT
TYPE OUT SELECTOR
M
F
L
A
X
CR
C
EMPTY
HSB TRANSFER
UNISERVO POWER
ON
OFF
MEM ADR SETUP
I ERROR CLEAR
O ERROR CLEAR
COMMA BRK PT
CLEAR RC
RETAIN C
RETAIN INST
ABNORMAL CONDITION
ON TEST
ERROR DELETED
STATIC REGISTER
EMPTY FILL
HI TEMP
RETAIN INST OR C
REG TYPE OUT
CR INTLOCK
FILL MEM
CR TYPE-IN
SKIP TYPE OUT
TYPE OUT BREAK POINT
ONE ADD
ONE STEP
ONE OP
ONE INST
CONT
INPUT READY
OUTPUT READY
12TH DIGIT
INPUT ERROR
READ
FORWARD
READY
BACKWARD
WRITE
READ
FORWARD
READY
BACKWARD
WRITE
1
2
3
4
5
6
7
8
9
–
A
B
C
D
E
F
DIRECTION MEMORY
REVERSE
FIRST BLOCK
READ INT
WRITE INT
BLOCK SUB DIV
TAPE DENSITY
A TAPE SELECTOR
1
2
3
4
5
6
7
8
9
–
A
B
C
D
E
F
RELEASE
RELEASE

UNIVAC® — The FIRST Name in Electronic Computing Systems