BASIC PROGRAMMING

UNIVAC® I

Data Automation System

Remington Rand Univac
DIVISION OF SPERRY RAND CORPORATION

BASIC PROGRAMMING

UNIVAC® I

Data Automation System

Remington Rand Univac
DIVISION OF SPERRY RAND CORPORATION

Table Of Contents

PAGE

PAGE

chapter 1

Elements of the Univac Data Automation System

To determine the elements of a data processing system, we will 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 this basis of the form the clerk brings the inventory up to date by writing a new column in the ledger.

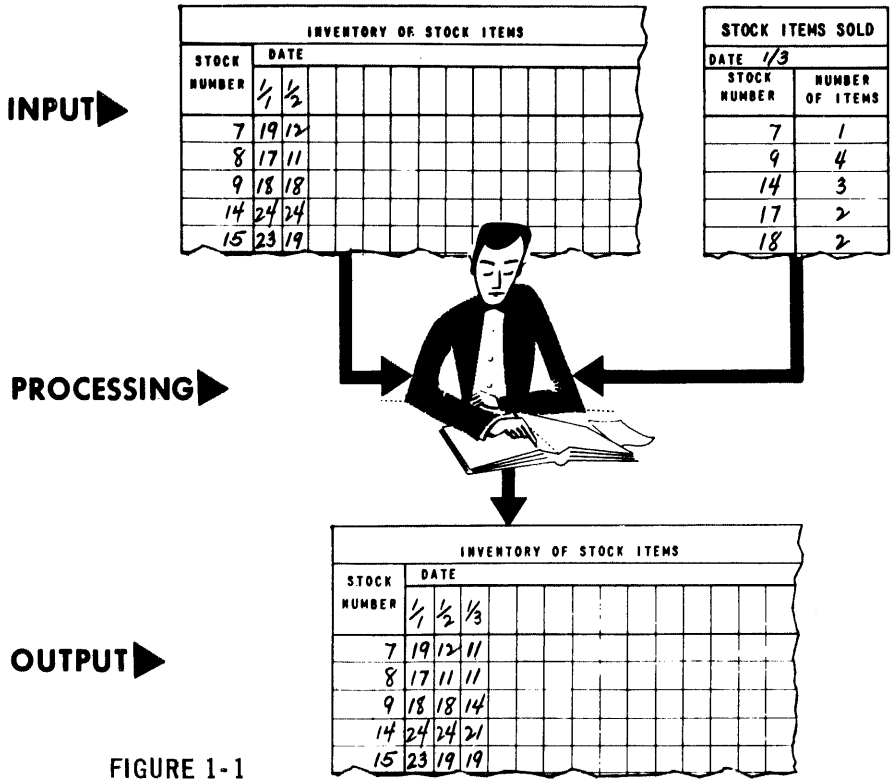


FIGURE 1-1

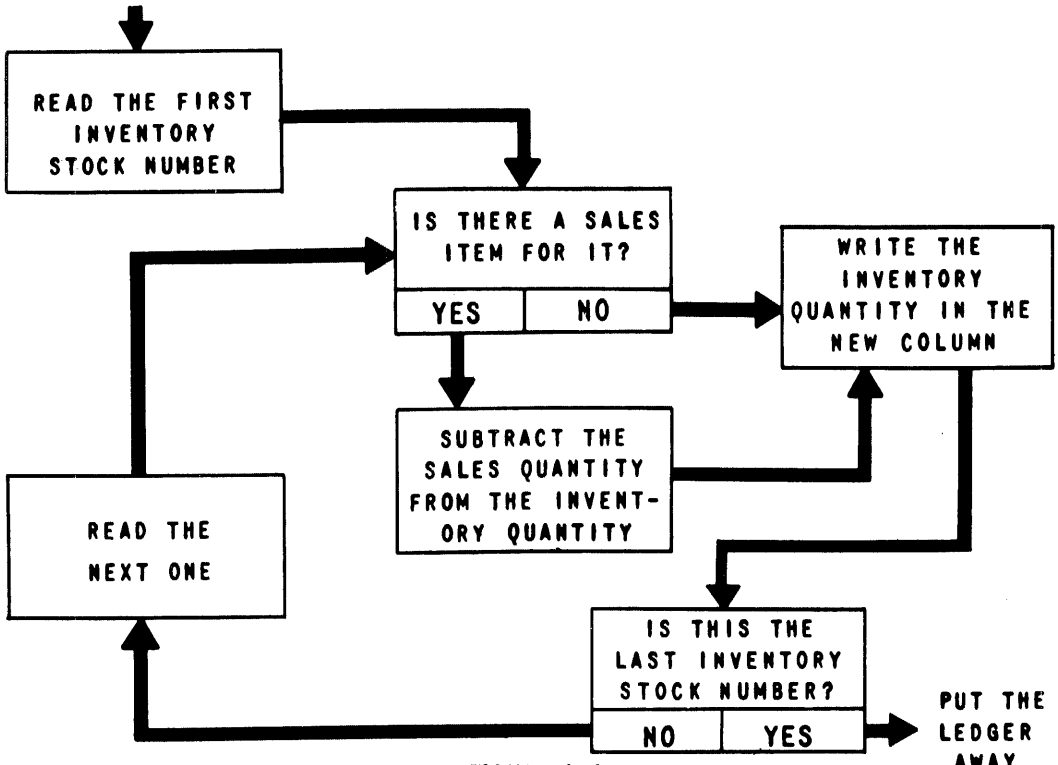


FIGURE 1-2

Thus, the clerk must be able to perform arithmetic;

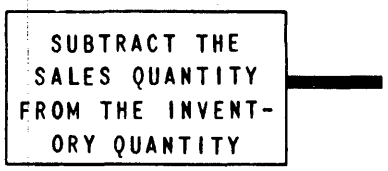


FIGURE 1-3

he must be able to make logical decisions;

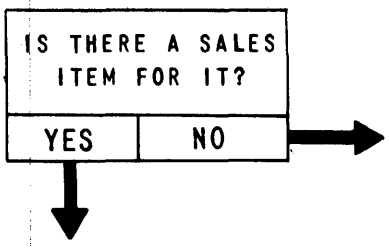


FIGURE 1-4

he must be able to remember information;

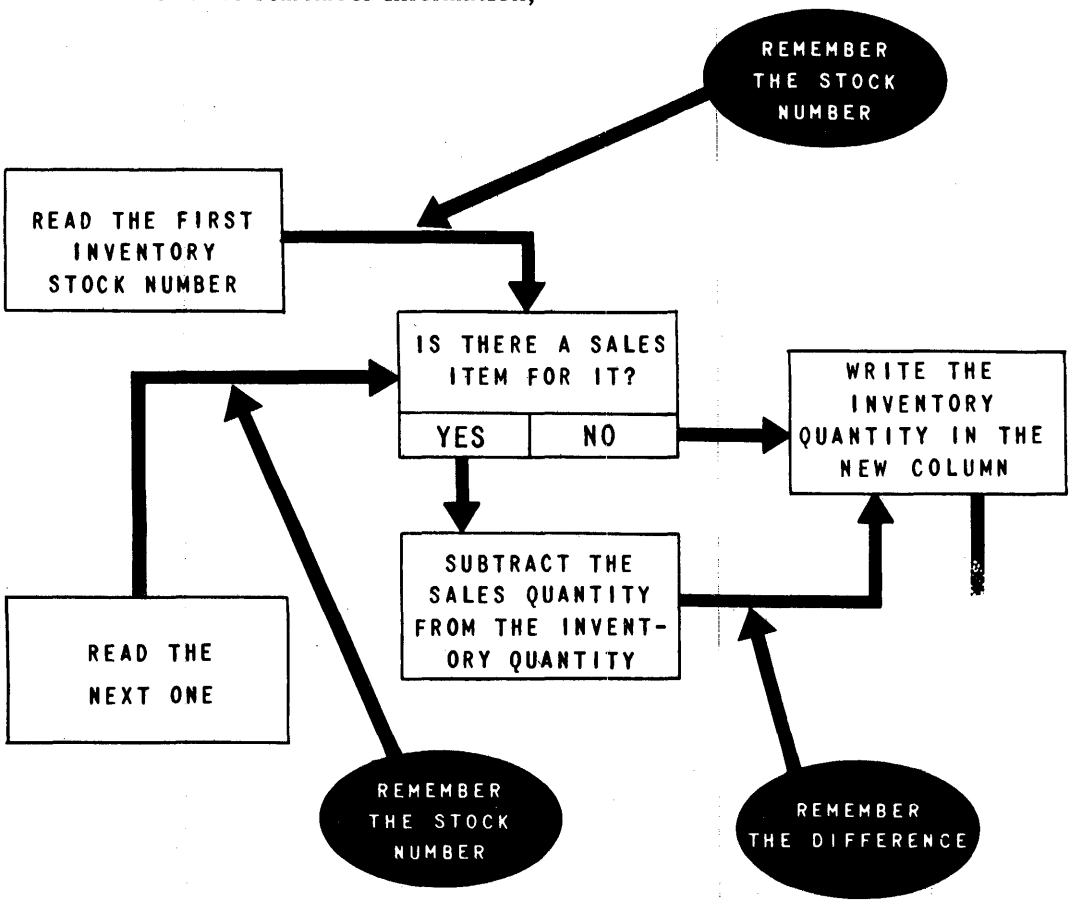


FIGURE 1-5

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

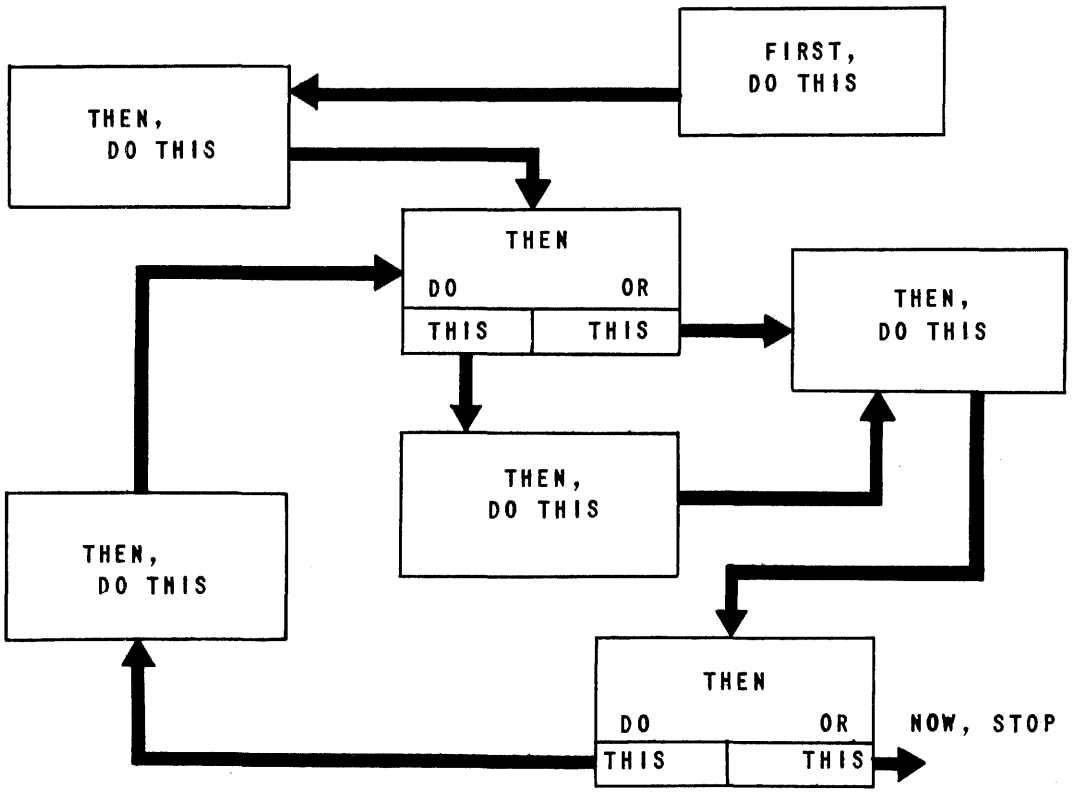


FIGURE 1-6

This example involves six elements.

1 Input	4 Memory
2 Arithmetic	5 Control
3 Logical Decisions	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.

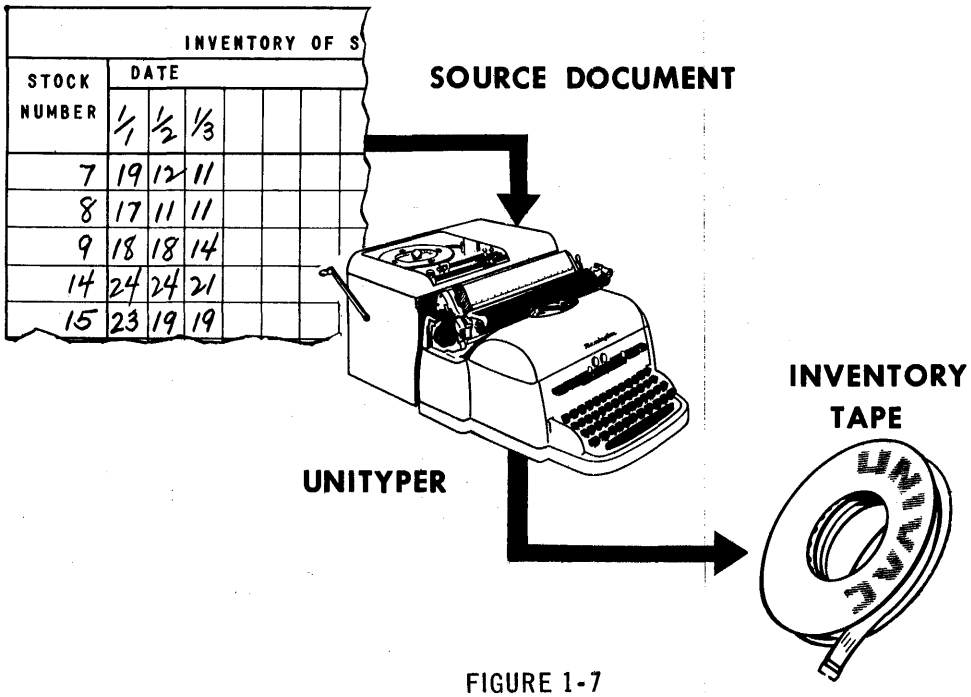


FIGURE 1-7

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.

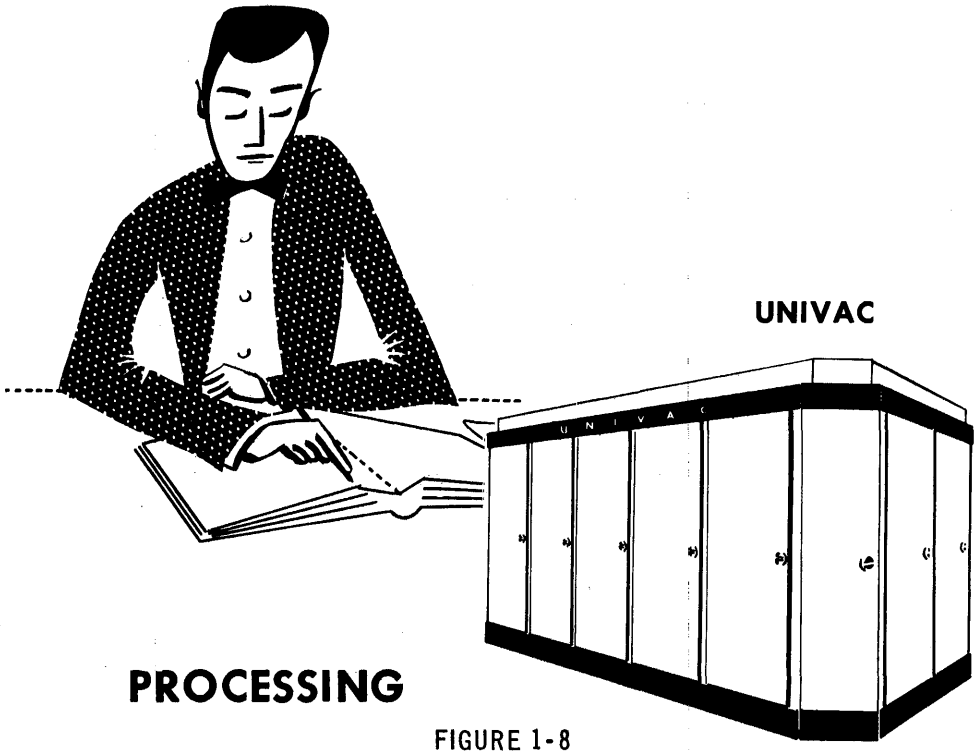


FIGURE 1-8

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

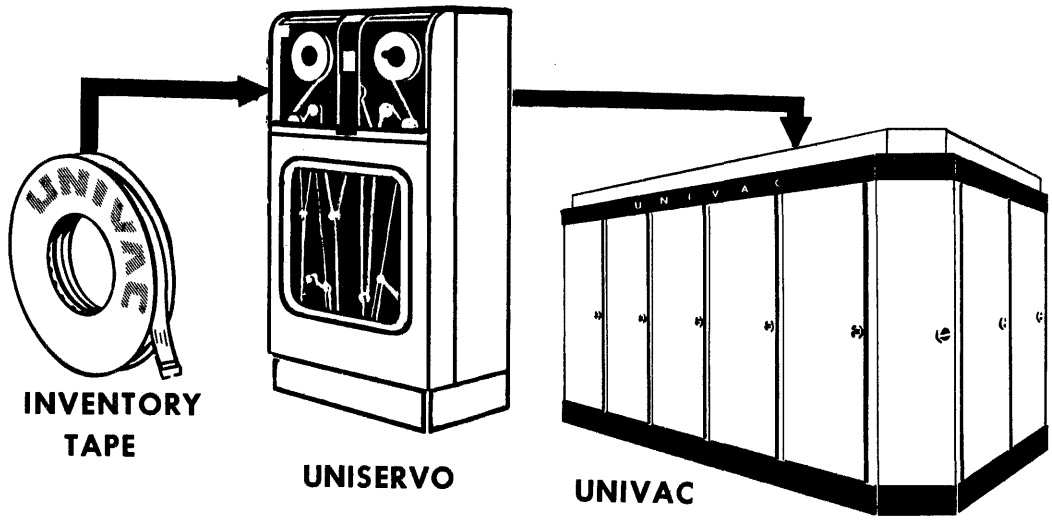


FIGURE 1-9

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 10 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

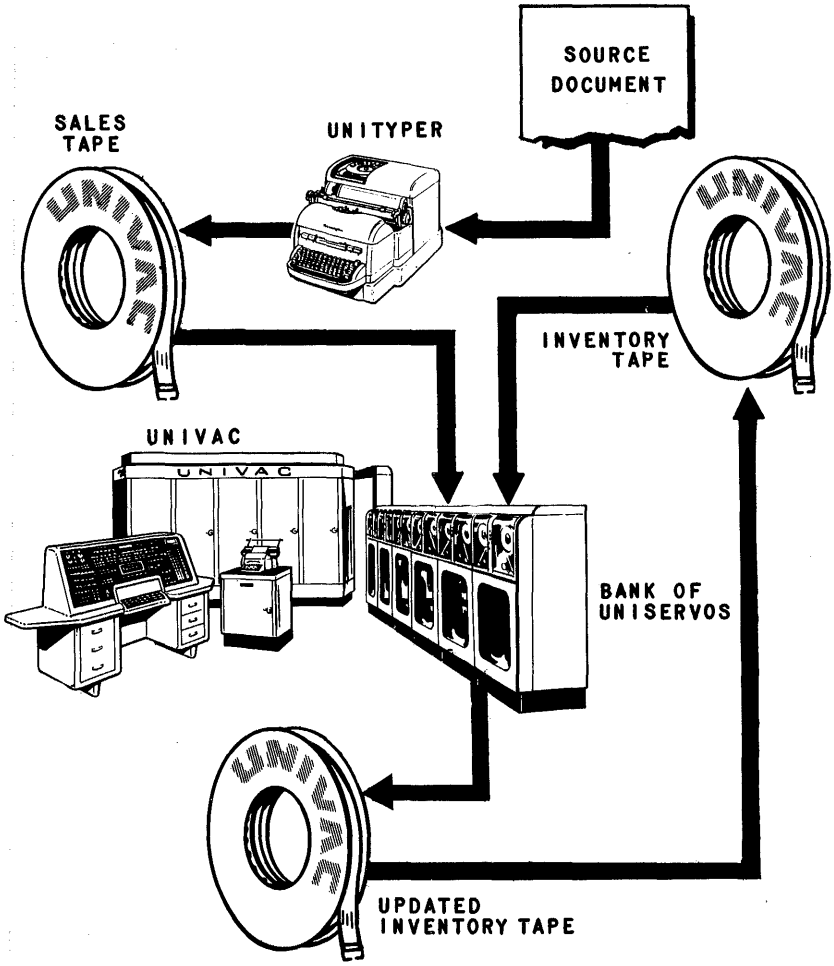


FIGURE 1-10

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 1-11

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

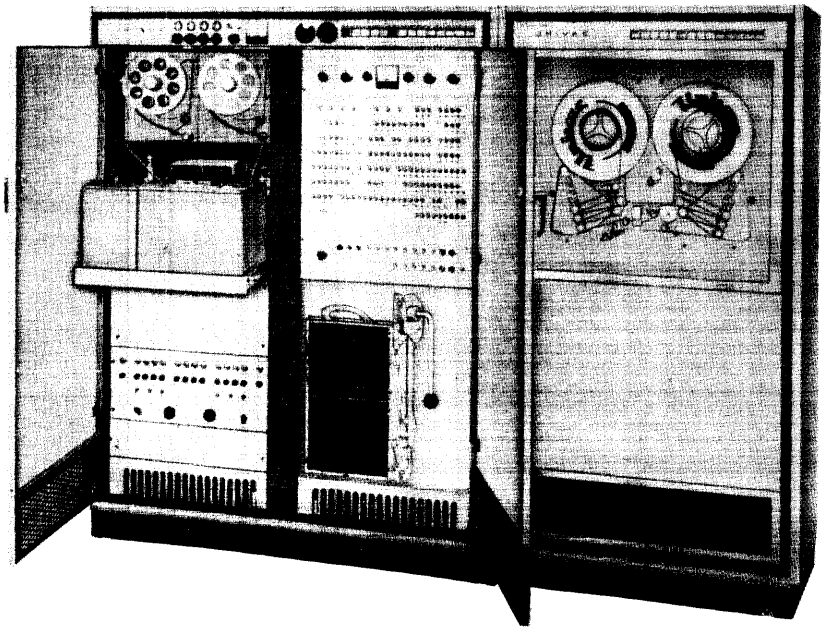


FIGURE 1-12 **PAPER-TO-MAGNETIC TAPE CONVERTER**

OUTPUT UNITS

The Univac High-Speed Printer.

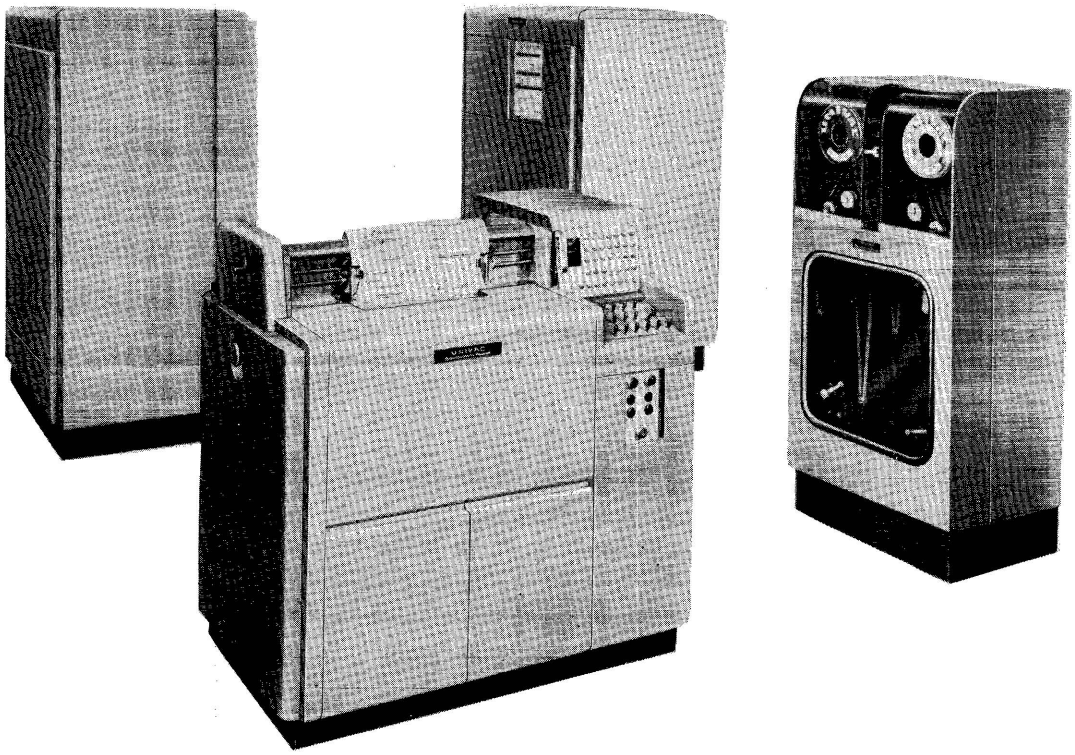


FIGURE 1-13 **HIGH-SPEED PRINTER**

The Univac Tape-to-Card Converter.

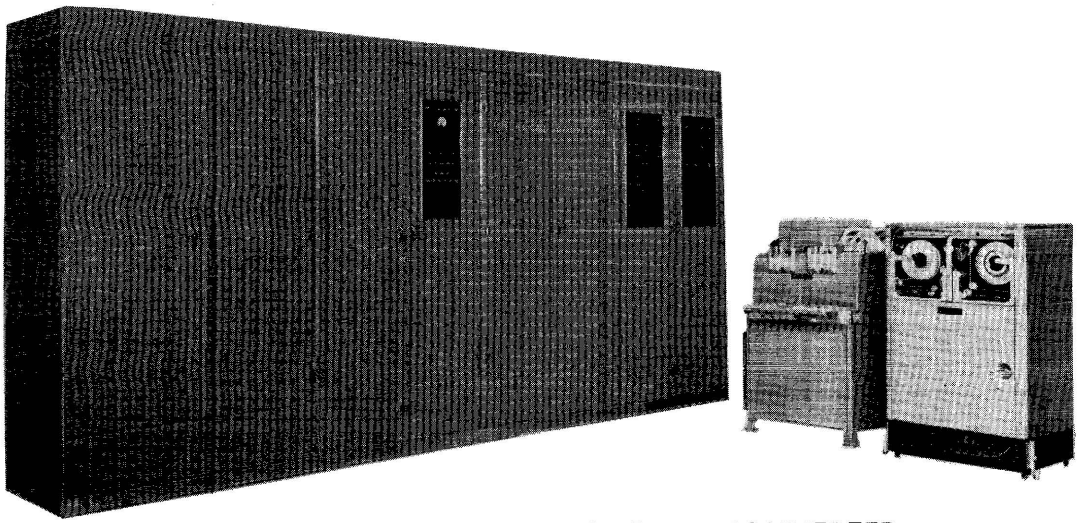


FIGURE 1-14 **TAPE -TO- CARD CONVERTER**

The Univac MTP converts magnetic to paper tape.

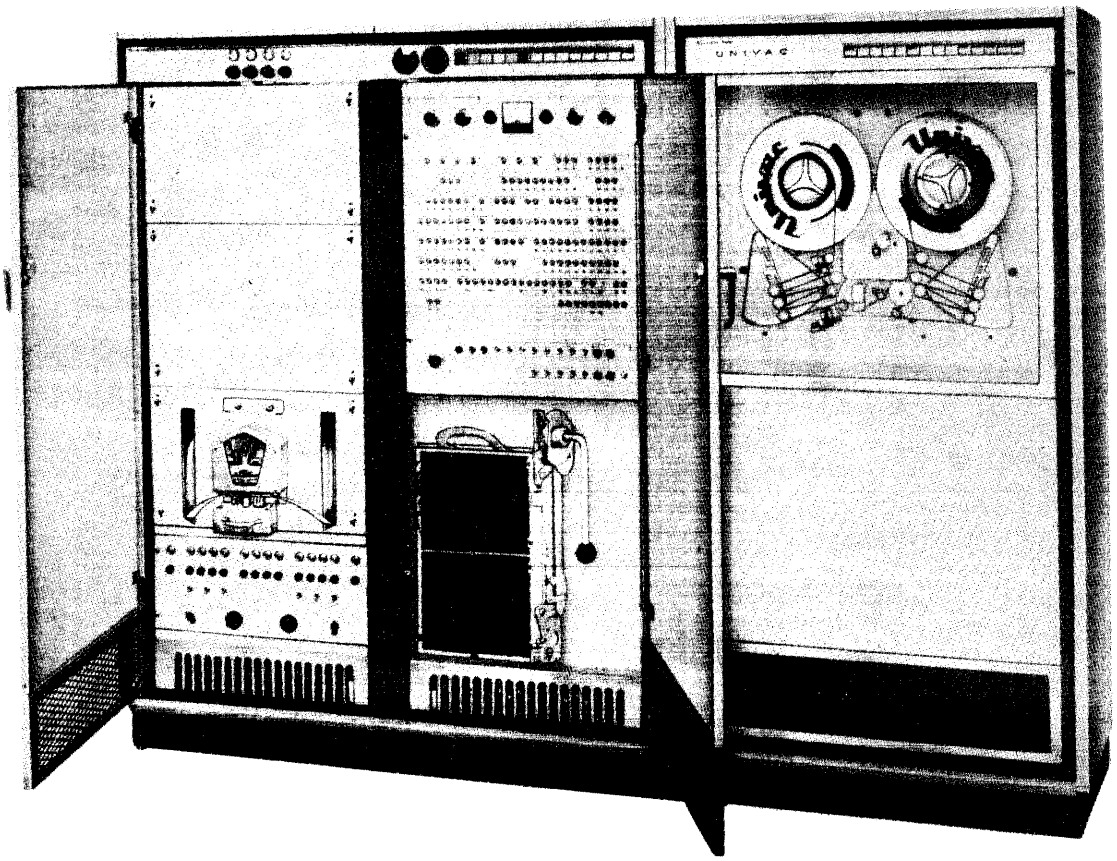


FIGURE 1-15 **MAGNETIC -TO-PAPER TAPE CONVERTER**

KEYBOARD INPUT OUTPUT

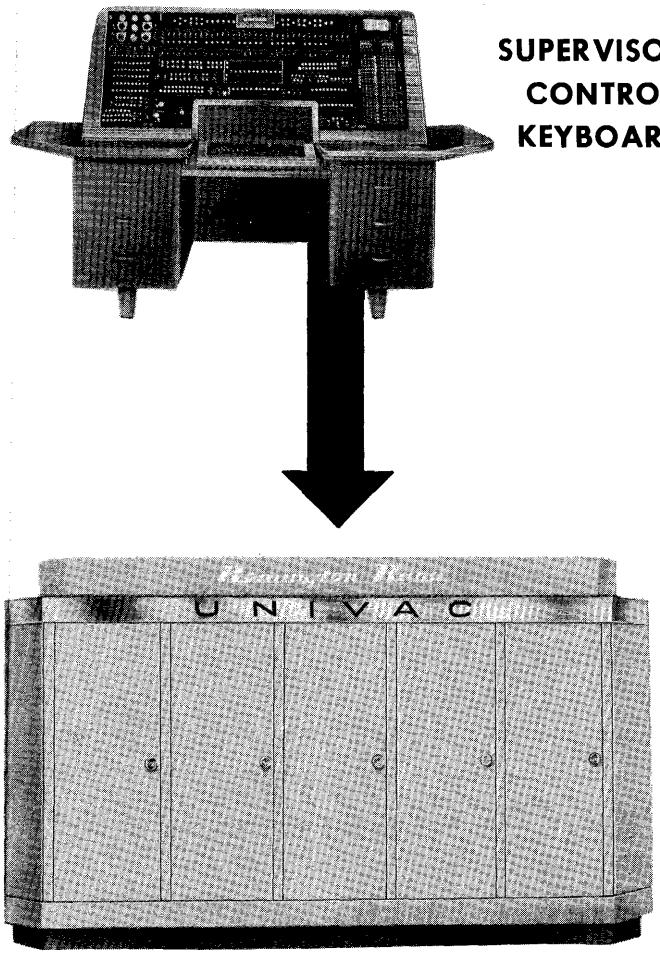
Besides using tape, the Central Computer can also accept and produce small volume data directly by means of a keyboard and a typewriter.

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.



SUPERVISORY CONTROL KEYBOARD

UNIVAC CENTRAL COMPUTER

FIGURE 1-16

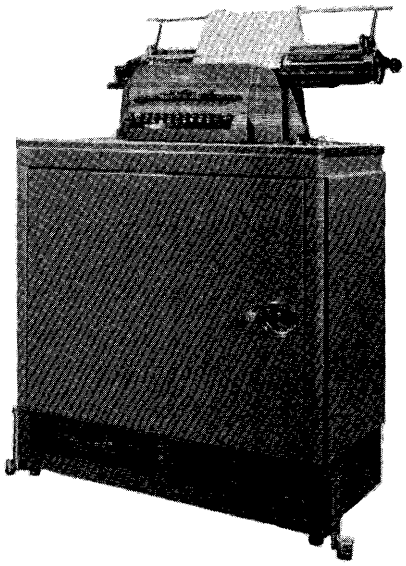


FIGURE 1-17
SUPERVISORY CONTROL PRINTER

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.

FIGURE 1-18

CHARACTERS

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

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.

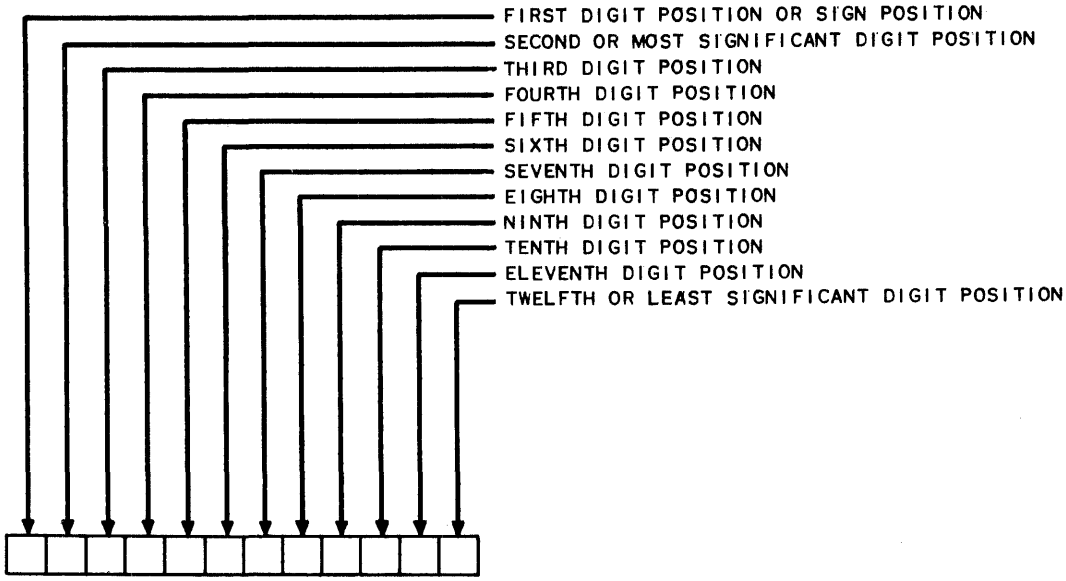


FIGURE 1-19

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.

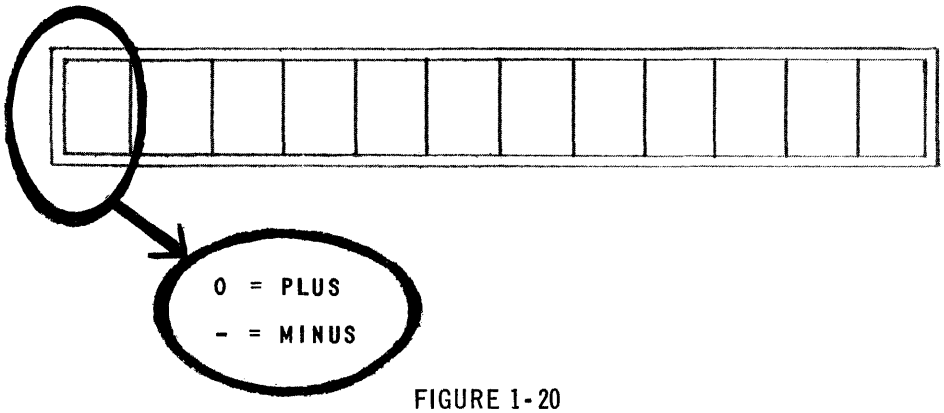


FIGURE 1-20

WORD AS A SIGNED QUANTITY

The memory size is 1000 cells. 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 000 to 999.

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

Figure 1-21 is a stylized version of the memory unit storing instructions, data and constants.

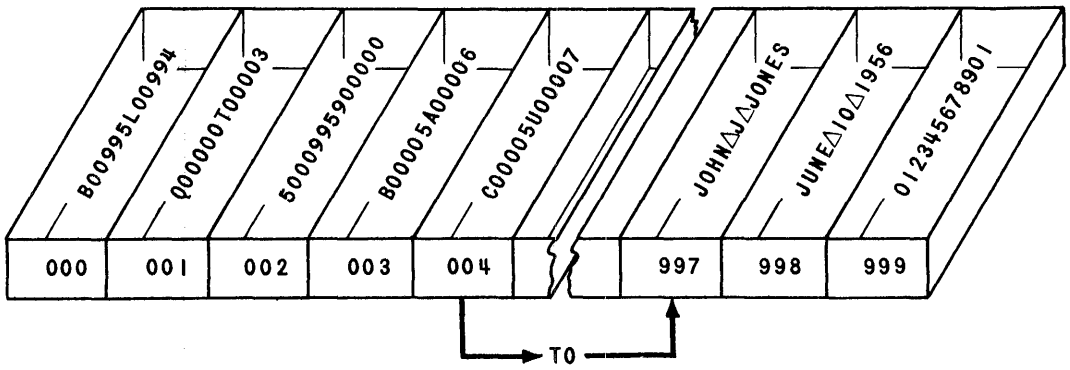


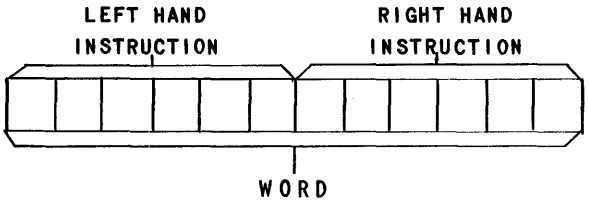
FIGURE 1-21

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 1-22

INSTRUCTION PAIR



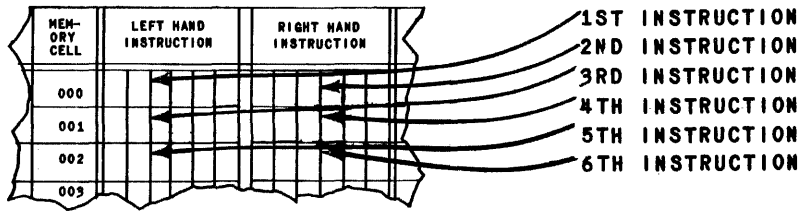
The function of the control unit is to select, in the proper sequence, each instruction in the memory, interpret it and execute it. Instructions are selected in pairs, one word, at a time. The left hand instruction (LHI) is executed, and then the right hand instruction (RHI).

The selection of instruction pairs is performed in a sequential manner. That is, if the instruction pair just executed is in cell 019, the next pair to be executed is in cell 020.

Initially the control unit begins the sequential execution of instruction pairs with the pair in cell 000. Thus, to have instructions executed in sequence, it is only necessary to represent the first instruction in the LHI of the word in cell 000; the second in the RHI of the word in cell 000; the third in the LHI of cell 001; and so on.

CONTROL

FIGURE 1-23



THE ARITHMETIC UNIT

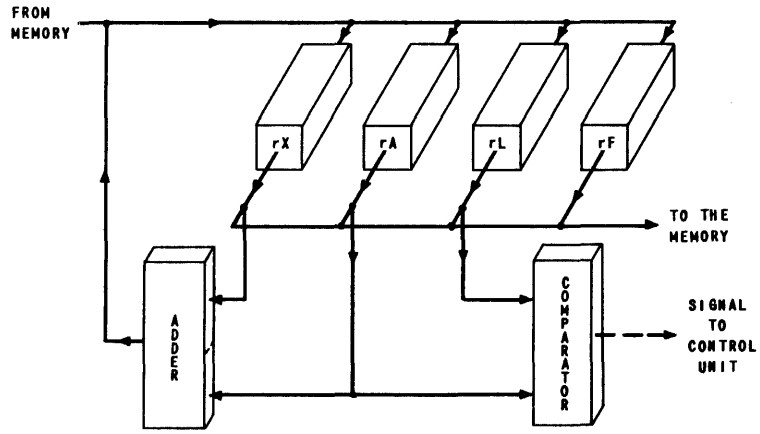
The arithmetic unit has characteristics in common with a desk calculator in that it contains an adder to produce the sum or difference of two words, a multiplier the product, and a divider to produce their quotient. In addition, to enable the Central Computer to make logical decisions, the arithmetic 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 1-24 is a stylized version of a portion of the arithmetic unit.

ARITHMETIC UNIT

FIGURE 1-24



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 V and Y will be described in detail in a later chapter).

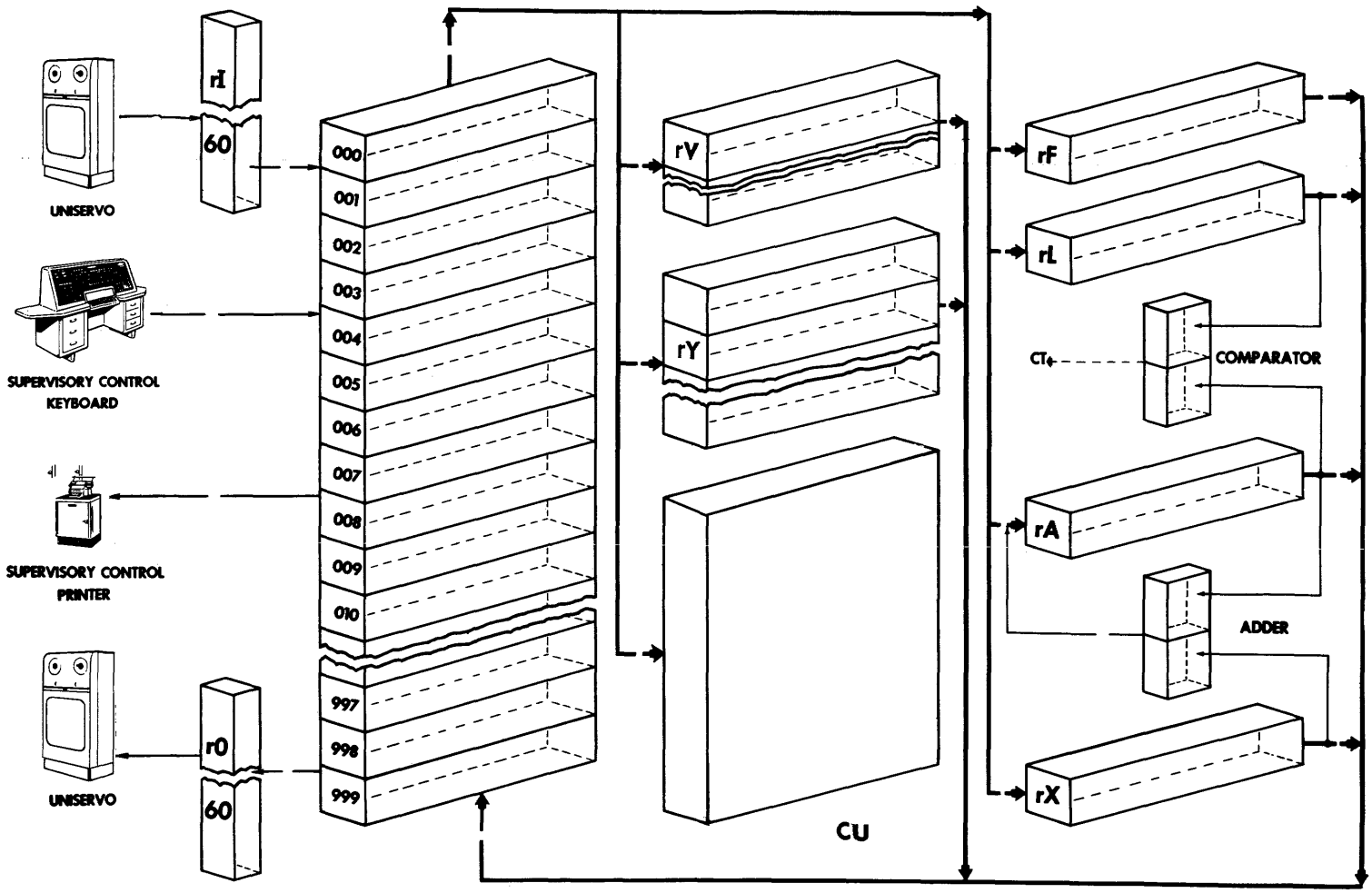

UNISERVO
rI
60
SUPERVISORY CONTROL
KEYBOARD
SUPERVISORY CONTROL
PRINTER
UNISERVO
rO
60
000
001
002
003
004
005
006
007
008
009
010
997
998
999
rV
rY
CU
rF
rL
CT
COMPARATOR
rA
ADDER
rX

FIGURE 1-25

chapter 2

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.

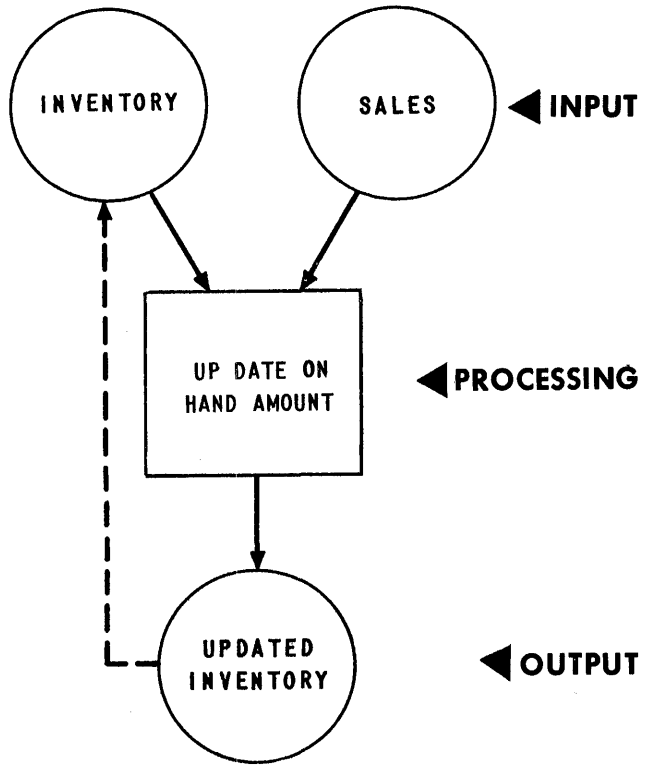
2. Logical Analysis - The analysis of the processing into a sequence of "small" logical steps.

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

PROCESS CHARTING

Figure 2-1 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 2-1

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.

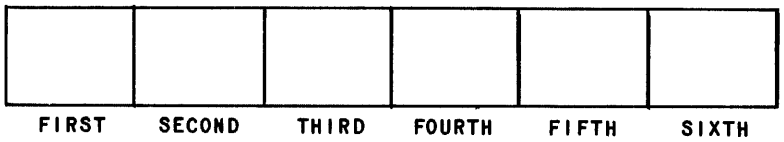
1. Select one quantity.
2. Add the second quantity to the first.
3. Record the sum.

If one quantity is in cell 880; the other, in 881; and if the sum is to be stored in cell 882; the instructions to cause the computer to do the above operations might be:

1. BRING 880
2. ADD 881
3. CLEAR 882

where BRING, ADD and CLEAR are code for the operations to be done; and 880, 881 and 882, 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 2-2

The first and second instruction digits indicate what operation is to be performed; the fourth through sixth digits, the address of the word affected by the operation. The third digit is normally a zero. (This digit is ignored in the execution of the instruction.)

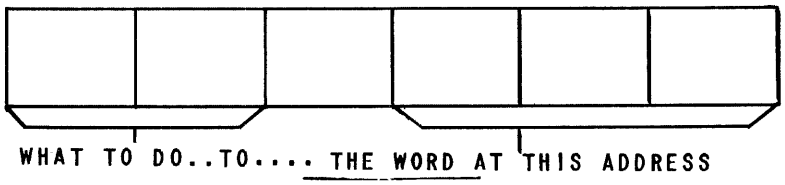


FIGURE 2-3

The instruction

500880

tells the central computer to perform the operation indicated by "50" on the word in cell 880.

ARITHMETIC INSTRUCTIONS - LIST A

An "m" is used to symbolize the fourth 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)$\longrightarrow$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)$\longrightarrow$rA, rX	Bring

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

INSTRUCTION	OPERATION	MNEMONIC
COm	(rA)$\longrightarrow$m; 0$\longrightarrow$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 880 is to be transferred to cell 881, the sequence of instructions might be

B00880 C00881

INSTRUCTION	OPERATION	MNEMONIC
HOm	(rA)$\longrightarrow$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.

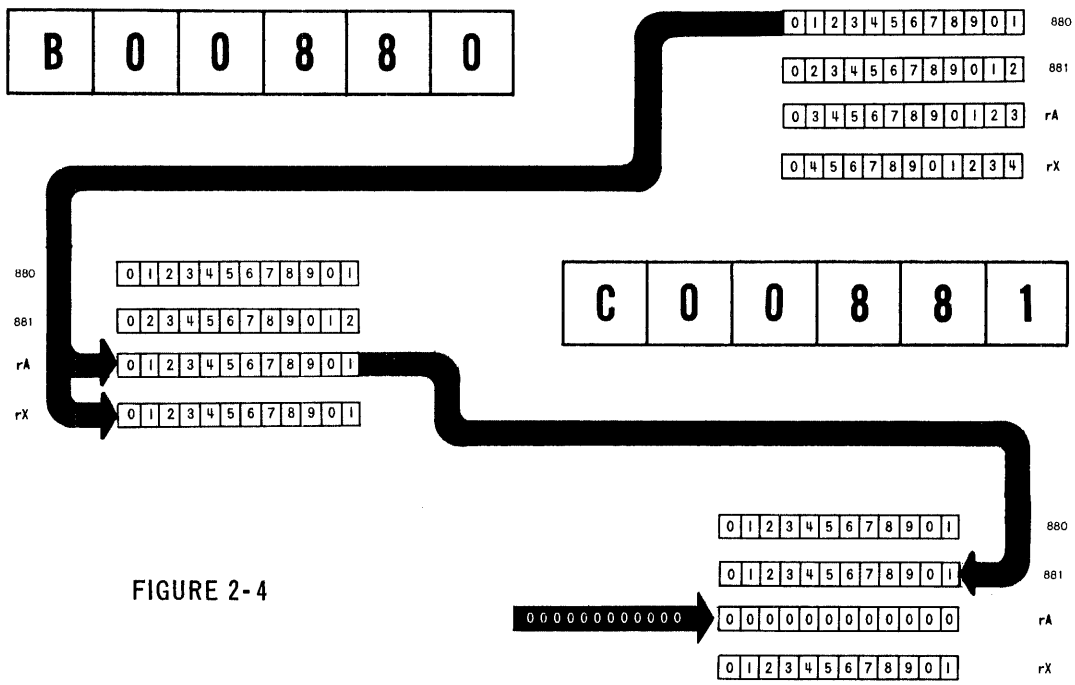


FIGURE 2-4

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 880 are to be duplicated in cells 881, 882 and 883, the instructions might be

	B00880	H00881
	J 00882	C00883
or:	B00880	J 00881
	J 00882	J 00883
or:	B00880	H00881
	H00882	H00883

etc.

INSTRUCTION	OPERATION	MNEMONIC
AOm	(m)⟶rX; (rA) + (rX)⟶rA	Add
	Add (rA) and (m), and transfer the sum to rA.	

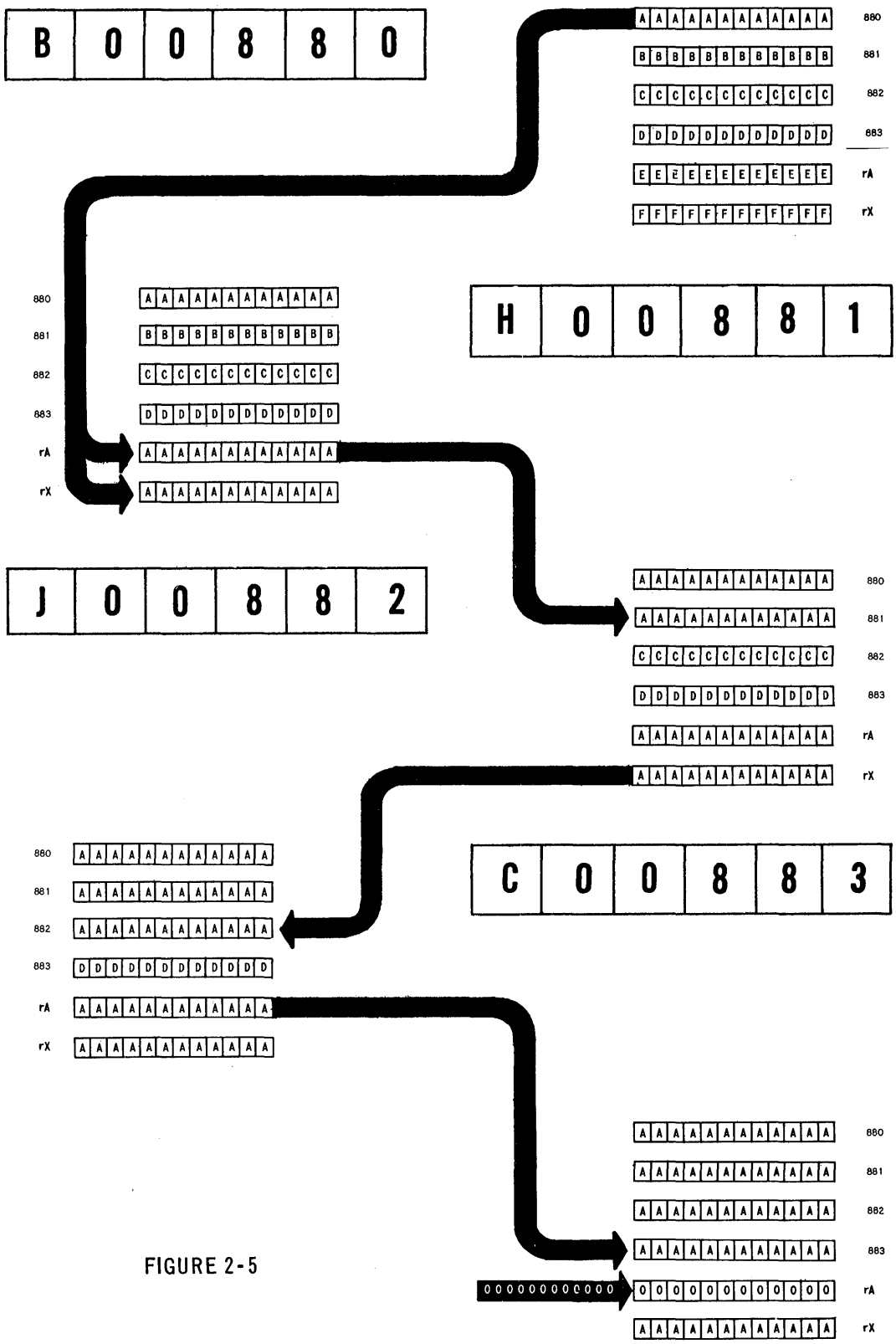

B 0 0 8 8 0
H 0 0 8 8 1
J 0 0 8 8 2
C 0 0 8 8 3
880
881
882
883
rA
rX
000000000000

FIGURE 2-5

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 880 to the contents of cell 881 and store the sum in 882, the sequence of instructions might be

B00880 A00881
C00882

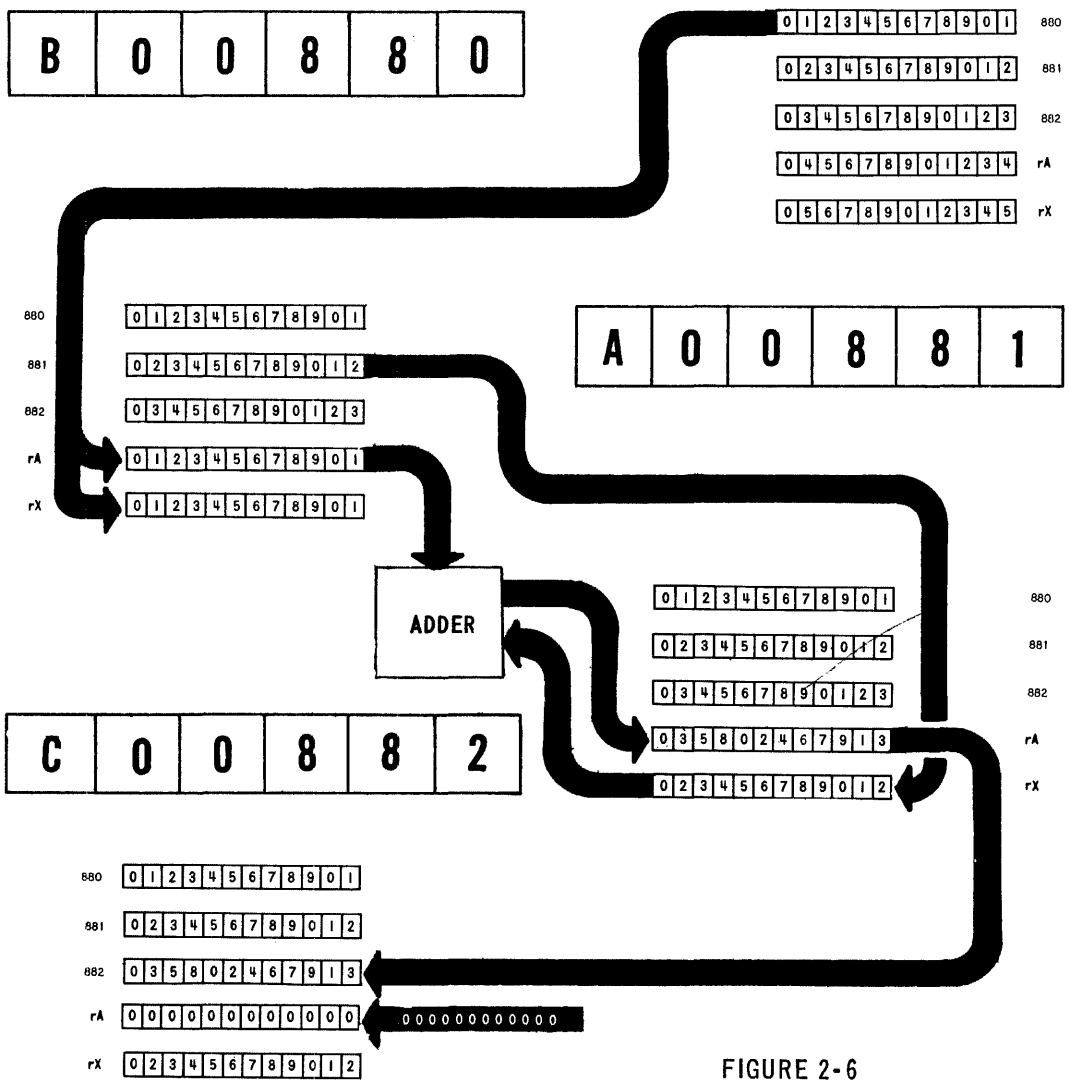


FIGURE 2-6

or B00880 A00881
H00882

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 880, the sequence of instructions to build up three times the quantity might be

B00880 X00000
X00000

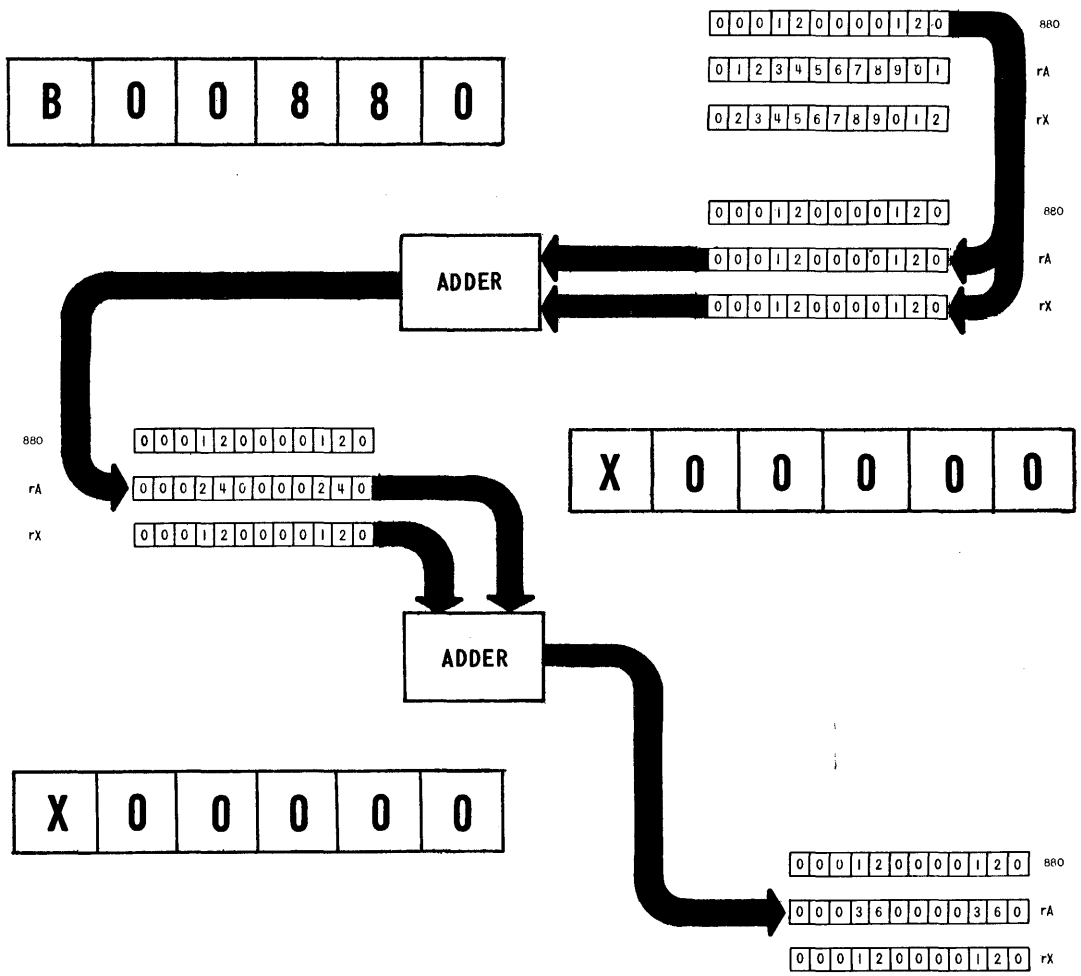


FIGURE 2-7

INSTRUCTION	OPERATION	MNEMONIC
SOm	$-(m) \longrightarrow rX;\ (rA) + (rX) \longrightarrow rA$	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.

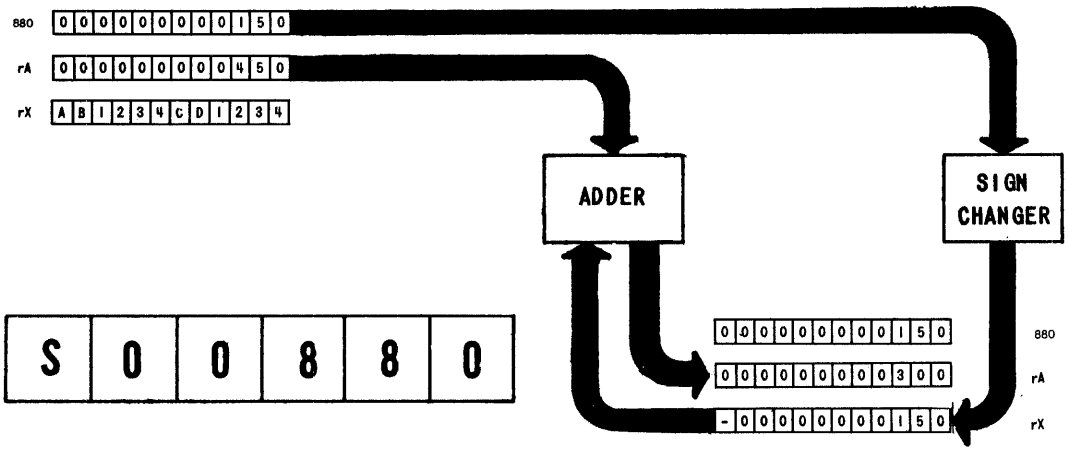


FIGURE 2-8

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

INSTRUCTION	OPERATION
50m	$(m) \longrightarrow SCP$

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

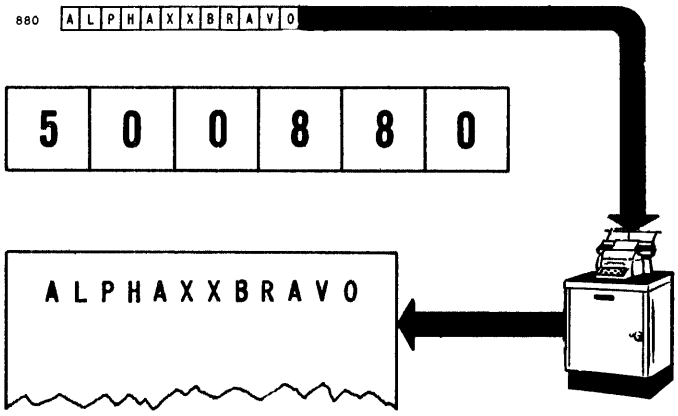


FIGURE 2-9

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 880, the ON ORDER quantity in cell 881, and the EXPECTED REQUIREMENTS for the next 60 days in cell 882. Print (on hand) + (on order) − (required). (Data will frequently be stored in memory starting at cell 880 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

000	READ	DATA	Read the data
001	B00880	A00881	Add the on order to the on hand
002	S 00882		Subtract the required from the sum
003	500883	C00883	Print the difference
		900000	Stop

The following is a description of the thinking that might have accompanied this coding.

Since the computer executes instruction pairs by starting with the pair in cell 000

and moving sequentially through the instruction pairs following, the instruction pairs should be stored in logical sequence, starting in cell 000. 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 000.

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 880 and 881. 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 001 should be:

B00880

At the completion of the B00880 instruction the on hand quantity will be in rA. To add the on order quantity to (rA), the instruction needed is

A00881

which should be the RHI of cell 001.

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 S00882 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 883 meets this requirement, and the instruction could be C00883. The execution of this instruction transfers the difference to cell 883. Then the execution of the instruction, 500883, will print the difference on SCP.

The last step is to stop the 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.

STUDENT EXERCISES

1. Reading the data stores a quantity in cell 880. Store the quantity in cells 881 and 882.

2. Reading the data stores two quantities in cells 880 and 881. Interchange the quantities.

3. Reading the data stores five receipt amounts in cells 880 - 884. Print the sum of the receipt amounts.

4. Reading the data stores four quantities, A, B, C and D, in cells 880 - 883. 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 880 - 883. If

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

print R.

ARITHMETIC INSTRUCTIONS - LIST B

INSTRUCTION	OPERATION	MNEMONIC
LOm	(m) $\longrightarrow$ rL, rX	Load

Transfer (m) to rL and rX, or <u>load</u> rL and rX with (m).

INSTRUCTION	OPERATION
KOm	(rA) $\longrightarrow$ rL; 0 $\longrightarrow$ rA

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

In executing the KOm instruction, the computer ignores m.

INSTRUCTION	OPERATION	MNEMONIC
POm	(m) rX; 3|(rL)| rE;	Precision
	(rL) x (rX) ⟶ rA [11 MSD], rX [11 LSD]	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)| ⟶ rE;	Multiply
	(rL) x (rX) ⟶ rA [11 MSD rounded], rX [11 LSD + .5]	

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;	Negative Multiply
	(rL) x (rX) ⟶ rA [11 MSD rounded], rX [11 LSD + .5]	

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 opera-

tion 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.

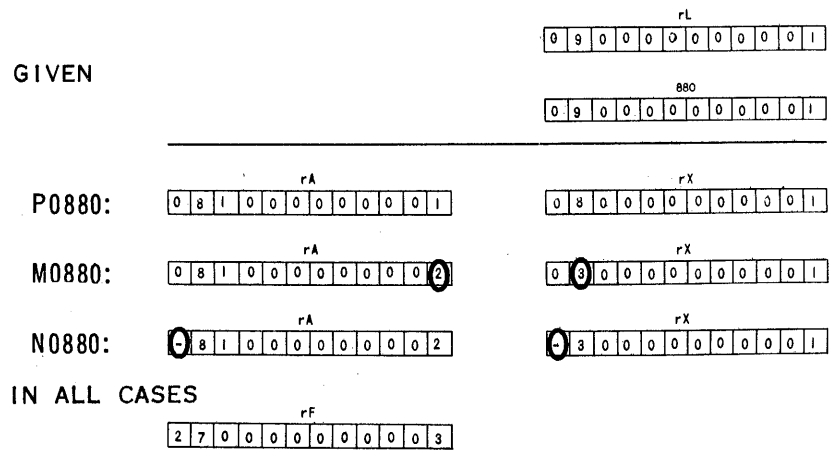


FIGURE 2-10

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 divide (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
101 012345678917

In executing D00101:

411522630566

12 digit quotient without sign

+000000000005

411522630571

Shift 1 place right and insert sign

rA 041152263057

rX 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	036000^500000	000000360^005
156.23	00156^2300000	000000015^623
$3756.28	03756^2800000	000000375^628

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 positions 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 = 0XXXXXXXXXXXX	m = 4
and	B = 0XXXXXXXXXXXX	n = 3

then

	AB = 0XXXXXXXXXXXX	m + n = 7
and	A÷B = 0XXXXXXXXXXXX	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

031200000000 = .312 (no assumption made)
031200000000 = .312 x 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 assumed 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 = 0XXXXXXXXXXX m = 9

and B = 0XXXXXXXXXXX n = 0

then

AB = 0XXXXXXXXXXX m + n = 9

and A÷B = 0XXXXXXXXXXX m − n = 9

STUDENT EXERCISES

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

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

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

4. Reading the data stores three quantities of form

   0QQQQQQQQQQQ

   in cells 880 - 882. Print the product of the quantities.

5. Reading the data stores

DATA	FORM	CELL
Quantity A	000AAAAAAAAA	880
Quantity B	000BBBBBBBBB	881
Quantity C	000CCCCCCCCC	882
Quantity D	000DDDDDDDDD	883

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	0IIIIIIII000	880
Number of Dependents	00NN00000000	881
Deductions other than for Dependents	000AAAAAA000	882

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

000000TTTTTT

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 can be stored in SR at any one time. Thus, SR is built with 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 three 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.

In Univac the extraction and execution of instructions is performed in four steps which are identified by the first four letters of the Greek alphabet:

STEP	DESCRIPTION
α	The right hand six digits of CC are duplicated in SR. The memory address section of SR now contains the address of the next instruction pair.
β	The effector circuits of SR now cause the contents of the memory cell as specified by the address section of SR to be duplicated in CR. A one is added in the least significant digit position of CC (digit position 12).
γ	The Left Hand Instruction now in CR is duplicated in SR, and being in SR causes the effector circuits to execute it: that is, interpret it as an instruction.
δ	The Right Hand Instruction in CR is duplicated in SR, and executed.

The computer automatically steps through the cycle and then after completing the δ step, begins on α. The important thing to note is that if CC = 000000000000 initially, the computer executes the Left Hand Instruction found in memory cell 000, then the Right Hand Instruction in that cell. Then, LHI of cell 001, RHI of 001, LHI of 002, RHI of 002, etc. Also note that instructions are executed only when they are in SR during stages γ or δ.

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	000000000 (CR) $\longrightarrow$ 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 three least significant digits of (CR) to the three 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 three 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.

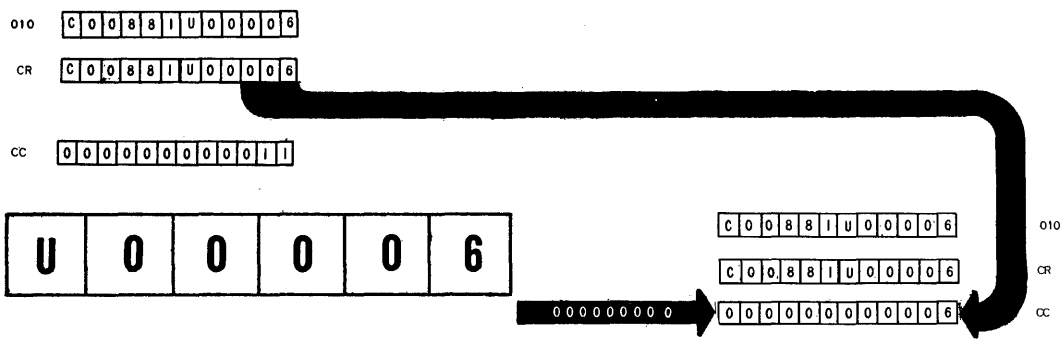


FIGURE 2-11

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.

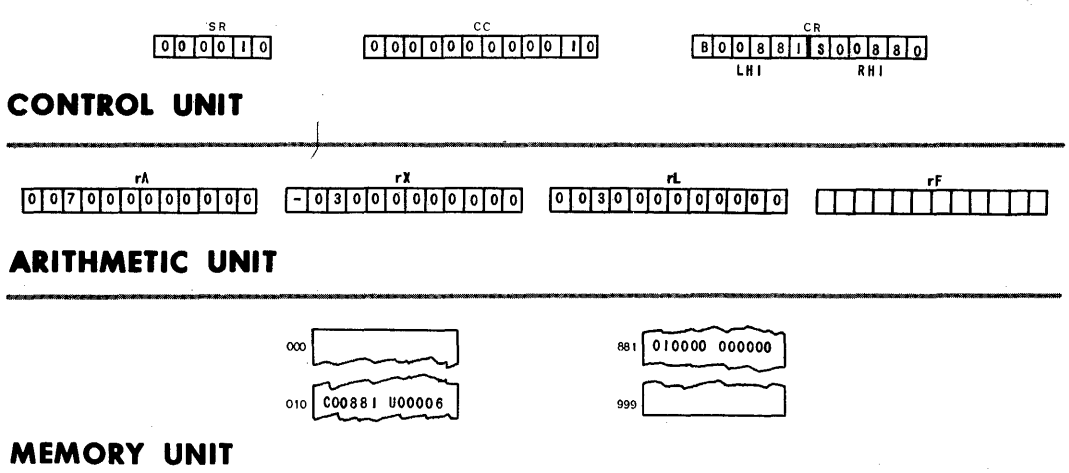


FIGURE 2-12

Assume that the computer has just completed beta time. During beta time the contents of cell 010, C00881U00006, were transferred to CR, and (CC) were increased by one.

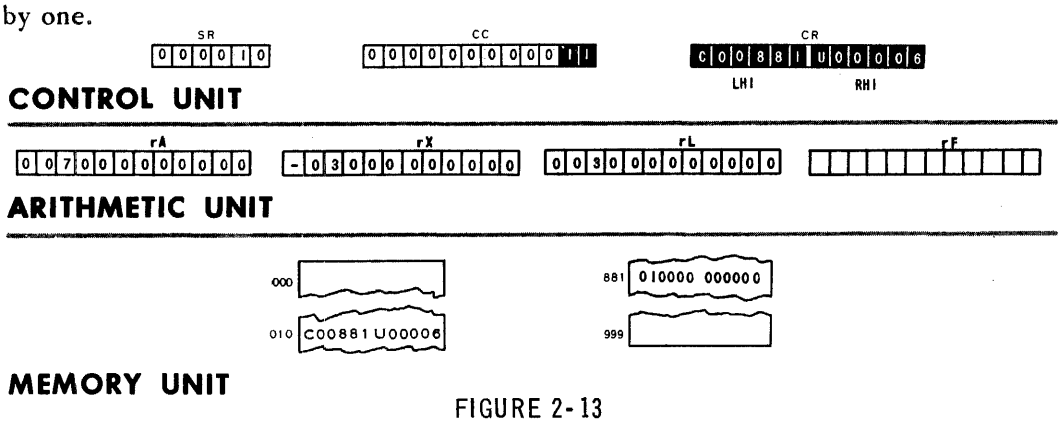


FIGURE 2-13

CR contains the current instruction pair, and the three least significant digits of (CC) specify that the next instruction pair is in cell 011. On gamma time C00081 is executed.

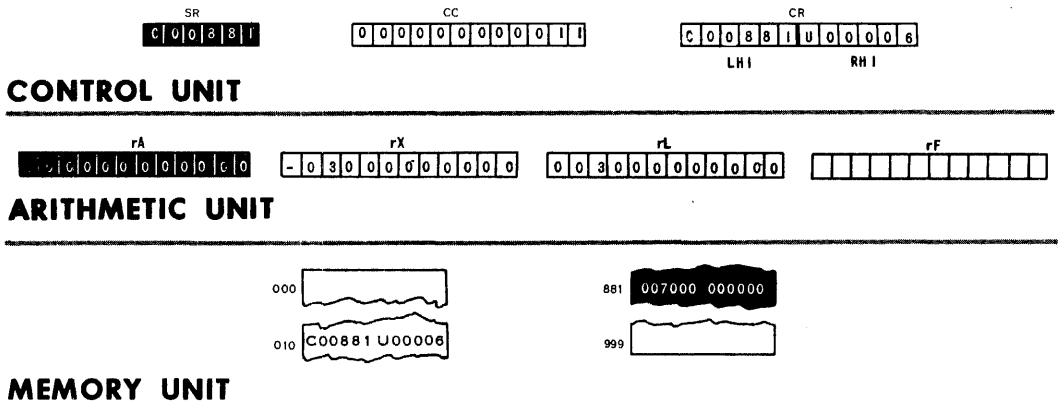


FIGURE 2-14

Delta time, U00006.

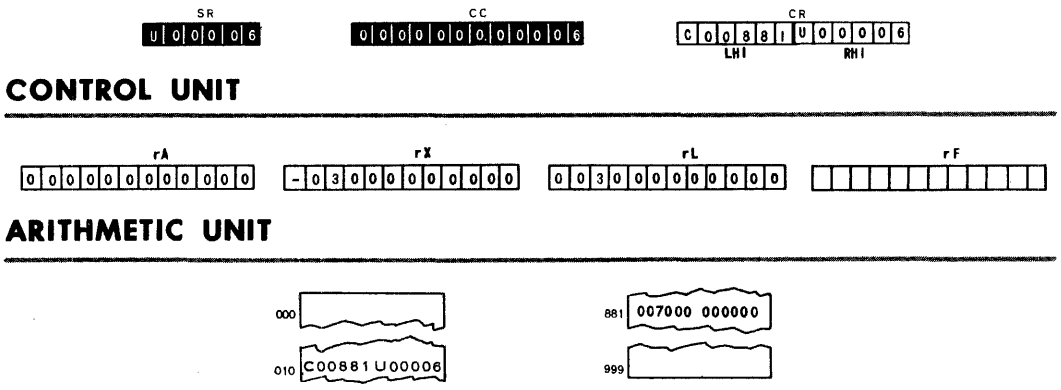


FIGURE 2-15

The three least significant digits of (CC) no longer specify that the next instruction pair is in cell 001, but instead specify that the pair is in cell 006. The computer's sequential operation has been broken, and control has been transferred to cell 006. With the instruction pair in cell 006 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 four stage cycle and must have some instruction to execute on gamma 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 1-18, 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	0AAAAAAAAAAAA	880
Delinquent Account Number	0DDDDDDDDDDDD	881

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

000	READ		Read the data
		DATA	
001	B00880		
		L00881	Is the account number equal to
002	⌣		the delinquent account number?
		Q00004	
003	500005		Print CREDIT GOOD.
		900000	Stop
004	500006		Print NO CREDIT
		900000	Stop
005	CREDIT		
		GOOD.	Constants
006	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.

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 three 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 003.

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

CREDITΔGOOD.

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 004, which can be specified by the QOm instruction. A 50m instruction and a 90m instruction in cell 004 complete the coding.

The next free cells are cells 005 and 006, which can be specified by the 50m instructions.

STUDENT EXERCISES

1. Reading the data stores:

DATA	FORM	CELL
Pay	000000PPPPPP	880
Deduction	00000000DDDD	881

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

2. Reading the data stores in cell 880, a charge in the form:

000000CCCCCC

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	880
On Hand	000000OOOOOO	881
Sold	000000SSSSSS	882
Minimum Required	000000RRRRRR	883

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	CELL
Quantity Ordered	0000QQQQQQ00	880
Unit Price	0PPPP0000000	881

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

chapter 3

Introduction to Flow Charts

EXAMPLE

Reading the data stores:

DATA	FORM	CELL
Days of Medical Absence	0AA000000000	880
Days of Allowable Medical Leave Remaining	0LL000000000	881
Hourly Rate of Pay	0RRRRR000000	882

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

00000000PPPP

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 storage.	5. Store medical absence in storage.	
6. Store zero in medical leave.	6. Reduce medical leave by medical absence.	
7. Multiply storage by eight.		
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.

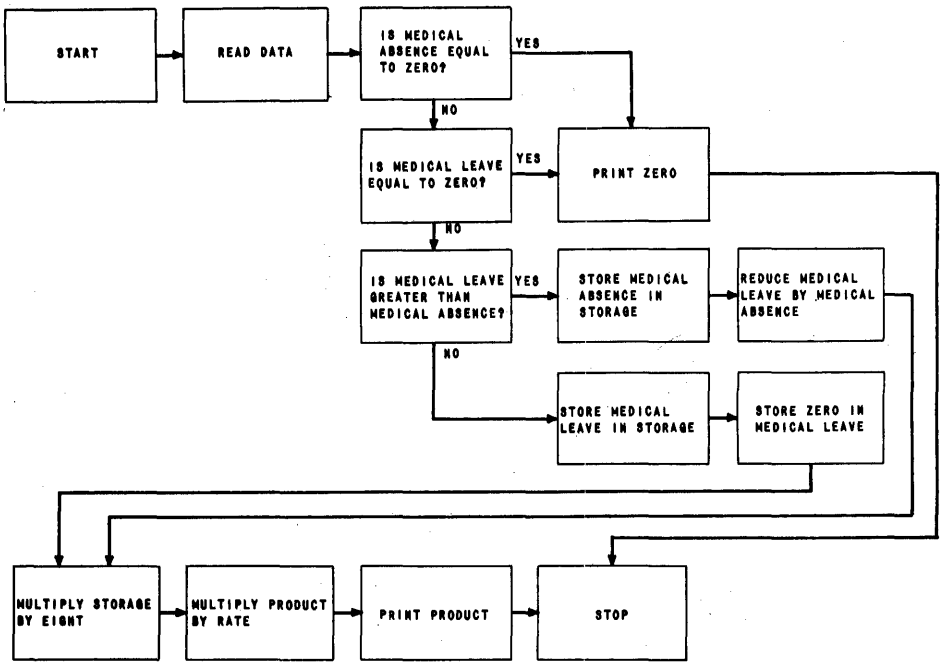


FIGURE 3-1

This solution to the problem of picturing the analysis is superior but would still result in a massive chart for a large problem. A further 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.

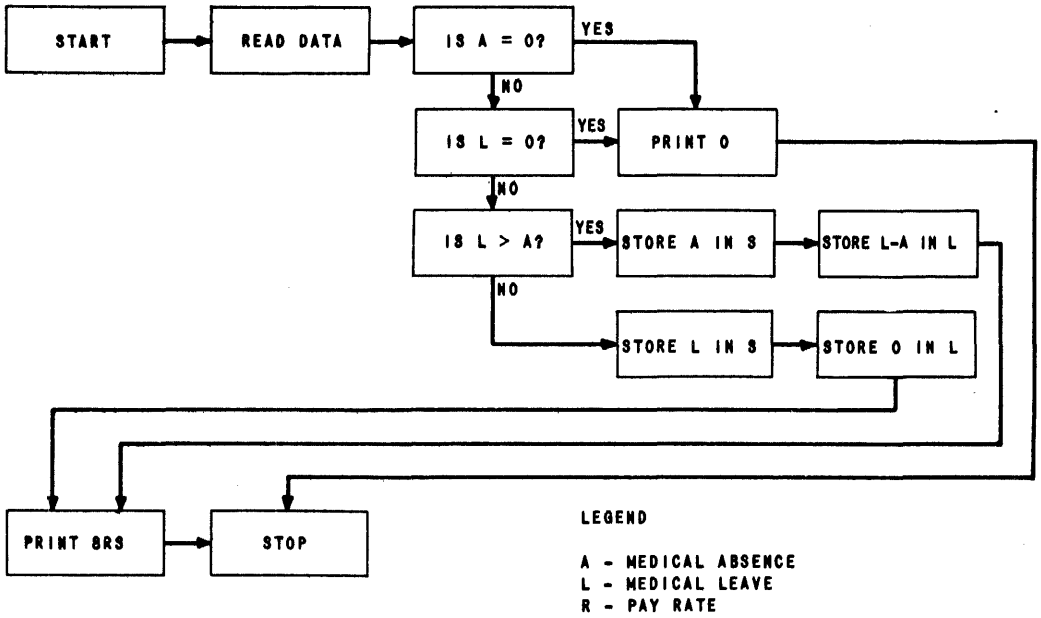


FIGURE 3-2

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

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

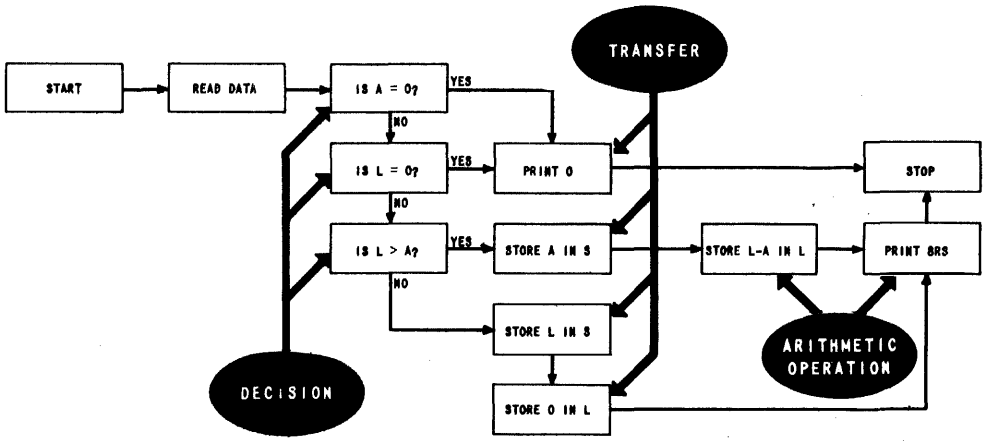


FIGURE 3-3

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

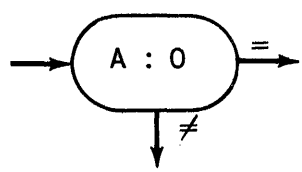


FIGURE 3-4

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

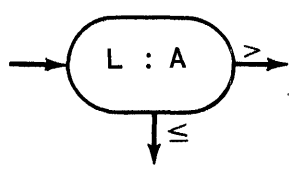


FIGURE 3-5

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(≤).

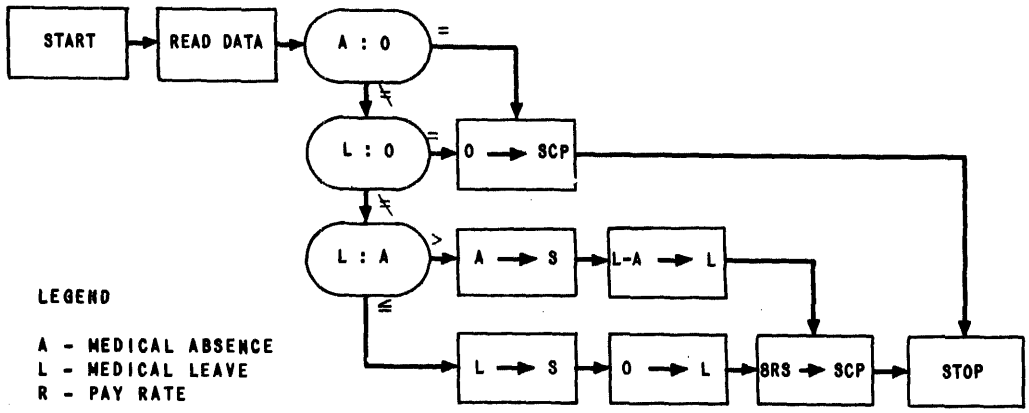


FIGURE 3-6

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,

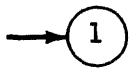


FIGURE 3-7

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

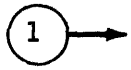


FIGURE 3-8

Thus,

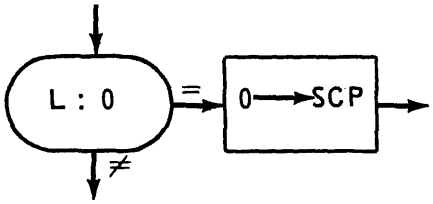


FIGURE 3-9

CODING

	000	READ		read data
			DATA	
	001	B00880		A : 0
			L00012	
	002			
			Q00011	
	003	B00881		L : 0
			Q00011	
	004	L00880		
			T00009	L : A
	005	K00000		L→S
			C00881	0→L
2	006	M00013		
			K00000	
	007	M00882		8RS→SCP
			C00883	
	008	500883		
			900000	Stop
	009	S00881		L − A→L
			C00881	
	010			
			U00006	
	011	500012		0→SCP
			900000	Stop
	012			
	013	000080		

ILLUSTRATIVE EXAMPLE

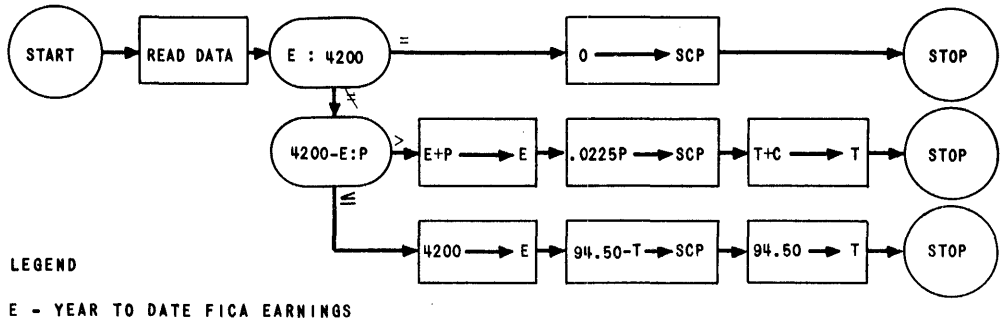
Reading the data stores

DATA	FORM	CELL
Year-to-Date FICA Earnings	000000EEEEEE	880
Year-to-Date FICA Tax	00000000TTTT	881
Current Pay	000000PPPPPP	882

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

00000000CCCC

FLOW CHART



LEGEND

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

FIGURE 3-12

CODING

000	READ		read data
		DATA	
001	B00880		E : 4200
		L00015	
002			
		Q00014	
003	B00015		4200 - E : P
		S00880	
004	L00882		
		T00010	
005	B00015		4200 → E
		C00880	
006	B00016		94.50 – T → SCP
		S00881	
007	C00883		
		B00016	94.50 → T
008	500883		
		C00881	
009	900000		Stop
010	B00880		E + P → E
		A00882	
011	C00880		.0225 P → SCP
		M00017	
012	H00883		
		A00881	T + C → T
013			
		U00008	

014	500018		0 ⟶ SCP
		900000	Stop
015			
		420000	
016			
		009450	
017	002250		
018			

STUDENT EXERCISES

Flow chart and code the following.

1. Reading the data stores a quantity of form

   ±QQQQQQQQQQ^

   in cell 880. 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^

   in cells 880 - 882.

   Print the smallest of the quantities.

3. Reading the data stores

DATA	FORM	CELL
Badge Number	NNNNNNNNNNNN	880
Bond Deduction	00000000DDDD	881
Cumulative Bond Deduction	0000000CCCCC	882
Bond Price	0000000 PPPPP	883

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	880
Quantity Sold	0000QQQQ0000	881
Unit Price	0PPPP0000000	882

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

0000000CCCCC

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

000000PPPPPP

in cell 880. The percentage tax is given in the following table.

PAY	TAX PERCENTAGE
$ 1 - 1499	1%
1500 - 2999	2%
3000 - 4499	3%
4500 - 5999	4%
6000 or over	5%

Deduct the tax, and print the net pay in form

000000NNNNNN

6. Reading the data stores

DATA	FORM	CELL
Year-to-Date Sales	0000SSSSSSSS	880
Year-to-Date Commission	000000CCCCCC	881
Current Sales	000000AAAAAA	882

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	880
Sales Quantity	000000SSSSSS	881
Minimum Requirements	000000RRRRRR	882

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

chapter 4

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

000	500003		
		A00005	
001	B00000		
		U00000	
002	C00000		
003	ΔΔELEC		Constants
		TRONIC	
004	ΔCOMPU		
		TER.ΔΔ	
005	000001		
		900000	

The execution of this coding will print:

ELECTRONIC COMPUTER.

and stop the computer.

First four stage cycle

Beta time - The contents of cell 000

500003000000

are transferred to CR.

Gamma time - The LHI

500003

is transferred to SR and executed, printing the contents of cell 003:

ELECTRONIC

Delta time - The RHI

000000

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

Second four stage cycle

Beta Time - The contents of cell 001

B00000A00005

are transferred to CR.

Gamma time - The LHI

B00000

is transferred to SR and executed, transferring the contents of the cell specified, cell 000,

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 four stage cycle, is now treated as data being processed by an instruction in SR.

Delta time p The RHI

A00005

is transferred to SR and executed, transferring the contents of cell 005 to rX, adding the contents of

	rA: 500003000000
and	rX: 000001900000

and transferring the sum

500004900000

to rA.

Third four stage cycle

Beta time - The contents of cell 002

C00000U00000

are transferred to CR.

Gamma time - The LHI

C00000

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

500004900000

to the cell specified, cell 000.

Delta time - The RHI

U00000

is transferred to SR and executed, transferring control to cell 000.

Fourth four stage cycle

Beta time - The contents of cell 000, which now contains the word

500004900000

are transferred to CR.

Gamma time - The LHI

500004

is transferred to SR and executed, printing

COMPUTER

Delta time - The RHI

900000

is transferred to SR and executed, stopping the computer.

On delta time of the second four 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 820, and 60 delinquent account numbers of form

DDDDDDDDDDDD

in cells 880 - 939. If the credit account number is equal to one of the delinquent account numbers print

ΔNOΔCREDIT . Δ

if not,

CREDITΔGOOD .

FLOW CHART

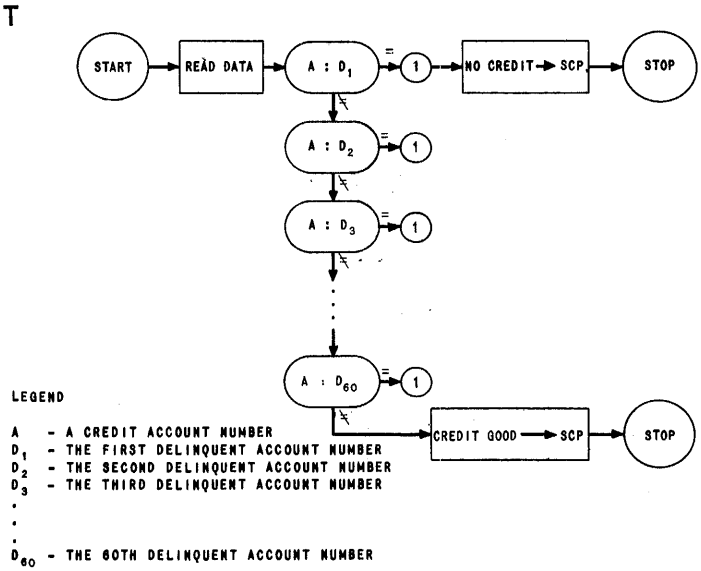


FIGURE 4-1

CODING

000	READ	DATA	read data
001	L00820		A : D_1
002	B00880	Q00063	
003	B00881	Q00063	A : D_2
004	B00882	Q00063	A : D_3
.			
.			
.			
061	B00939	Q00063	A : D_{60}
062	500064	900000	CREDIT GOOD → SCP stop
1 063	500065	900000	NO CREDIT → SCP stop
064	CREDIT	ΔGOOD.	constants
065	ΔNOΔCR	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 L00820, takes the form

B00XXXQ00063

where XXX 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.

000	READ	DATA	read data
001	B00880	L00820	Does the credit account number match the current delinquent account number?
002		Q00008	

```
003   B00001                          B0880L0820 → rA
                 ↺
004   ↺                               Take the next delinquent account number.
                 ↺
005   A00010                                add     000001 000000
                 C00001                                B0881  L0820 → 0001
006   ↺
                 U00001
-------------------------------------
007   ↺
                 ↺
008   500012                          NO CREDIT → SCP
                 900000               stop
-------------------------------------
009   ↺
                 ↺
010   000001
                 ↺                    constants
011   ↺
                 ↺
012   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 001 will eventually contain the instruction pair

B00939L00820

After each iteration the contents of cell 001 can be compared for identity with the above word. This comparison determines 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.

```
000   READ                            read data
                 DATA
001 [ B00880                          Does the credit account number match
                 L00820 ]             the current delinquent account number?
002   ↺
                 Q00008
-------------------------------------
```

```
003   B00001
                L00009
004
                Q00007
005   A00010
                C00001
006
                U00001
007   500011
                900000
008   500012
                900000
009   B00939
                L00820
010   000001
011   CREDIT
                ΔGOOD.
012   ΔNOΔCR
                EDIT.Δ
```

Is the current delinquent account number the last delinquent account number?

Take the next delinquent account number.

CREDIT GOOD → SCP
stop

NO CREDIT → SCP
stop

constants

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 61 the constant used to determine if all delinquent account numbers have been processed is

B00939L00820

In the following coding the constant is

B00940L00820

```
000   READ
                DATA
001 [ B00880
                L00820 ]
002
                Q00007
```

read data

Does the credit account number match the current delinquent account number?

003	B00001		
		L00008	Is the current delinquent account number
004	A00009		the last delinquent account number?
		Q00006	
005	C00001		Take the next delinquent account number.
		U00001	
006	500010		CREDIT GOOD → SCP
		900000	stop
007	500011		NO CREDIT → SCP
		900000	stop
008	B00940		
		L00820	
009	000001		
		↪	constants
010	CREDIT		
		ΔGOOD.	
011	ΔNOΔCR		
		EDIT.Δ	

The reason for the difference in constants is that, in the coding on page 61, 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 62, 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 61	In Coding on Page 62
1st	B00880L00820	B00881L00820
2nd	B00881L00820	B00882L00820
3rd	B00882L00820	B00883L00820
.	.	
.	.	
.	.	
58th	B00937L00820	B00938L00820
59th	B00938L00820	B00939L00820
60th	B00939L00820	B00940L00820

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:

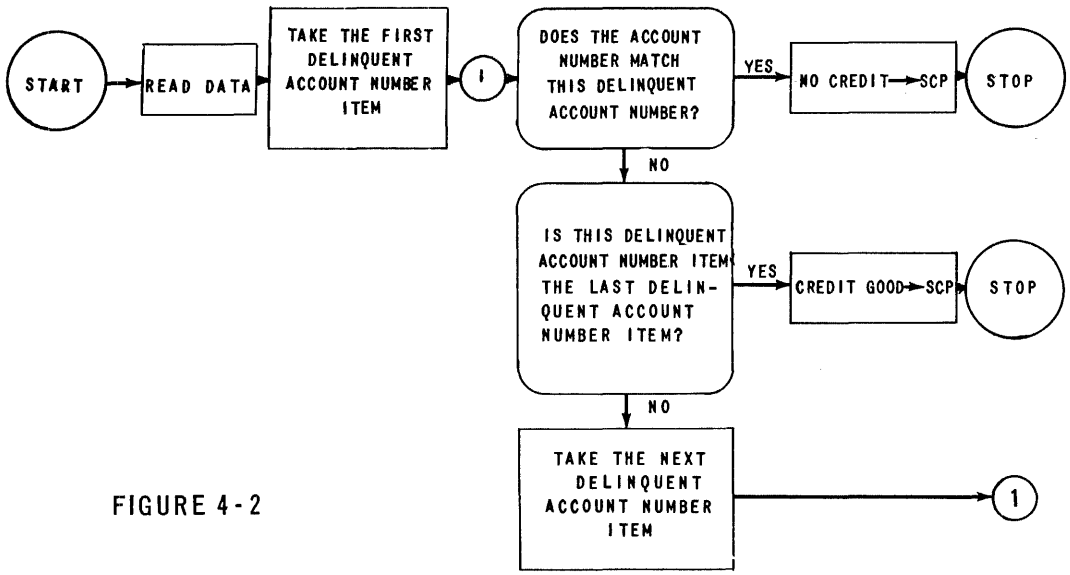


FIGURE 4-2

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

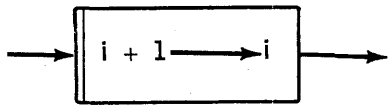


FIGURE 4-3

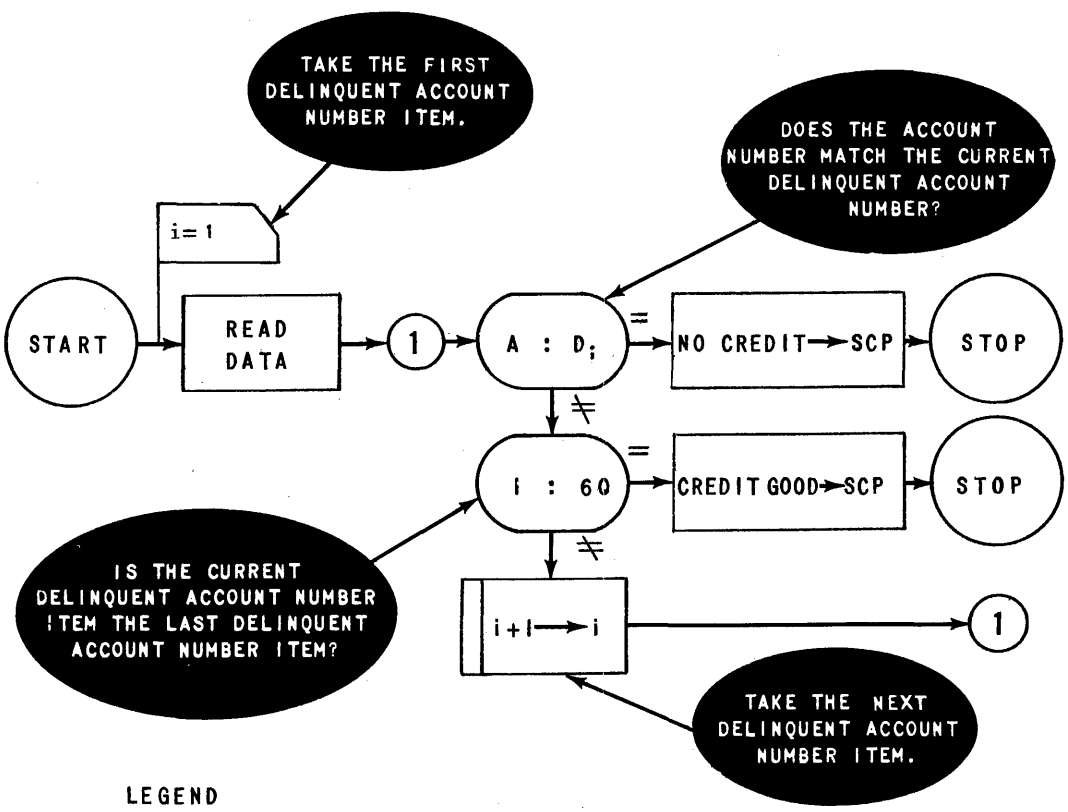
provides this sequence. The operation box has a double line on the left to distinguish it from an operation which processes data. The initial condition for the sequence is that i be equal to one so that $D_i = D_1$. Initial conditions of the processing are shown in an assertion flag placed immediately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

ILLUSTRATIVE EXAMPLE

Reading the data stores 60 receipt amounts of form

000000RRRRRR

in cells 880-939. Print the sum of the amounts.



LEGEND

A AN ACCOUNT NUMBER
D A SET OF DELINQUENT ACCOUNT NUMBER ITEMS
D_i THE iTH ITEM IN D, i = 1, . . . , 60

FIGURE 4-4

FLOW CHART

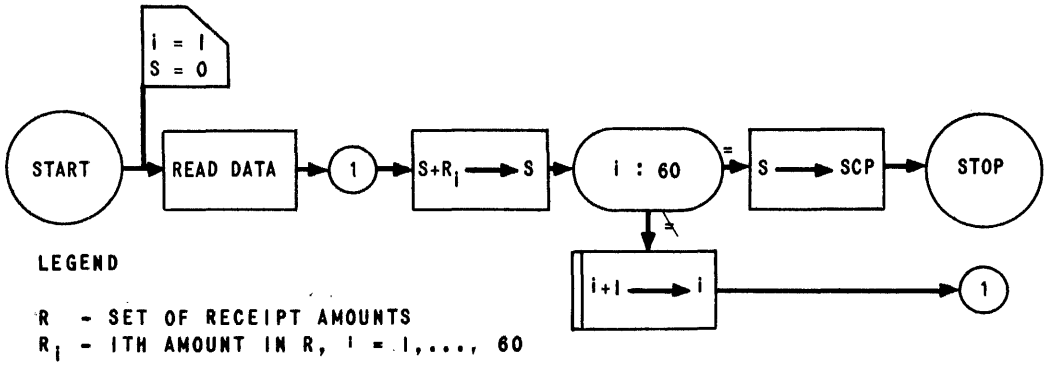


FIGURE 4-5

The following is a description of the thinking that might have accompanied the flow chart.

Flow chart the general processing

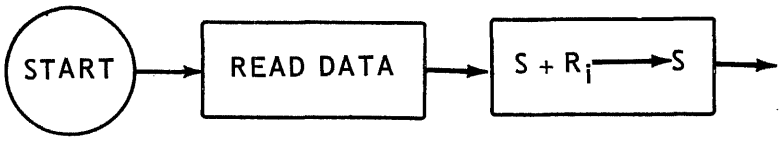


FIGURE 4-6

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

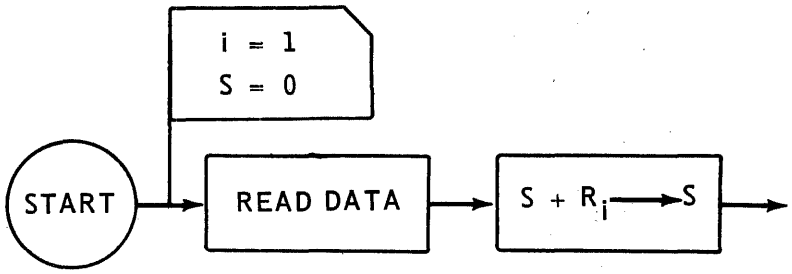


FIGURE 4-7

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

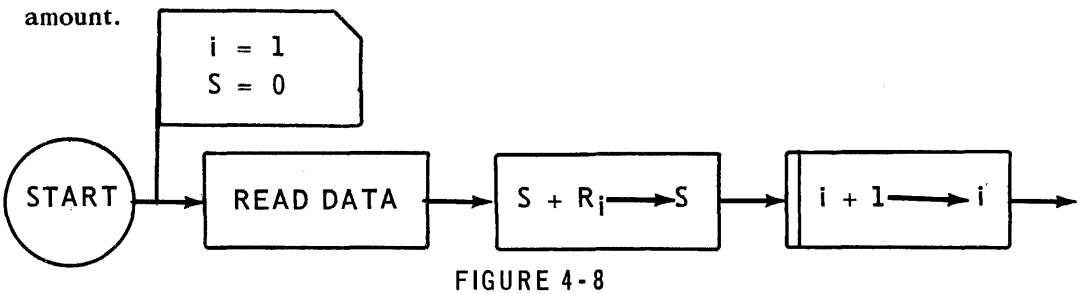
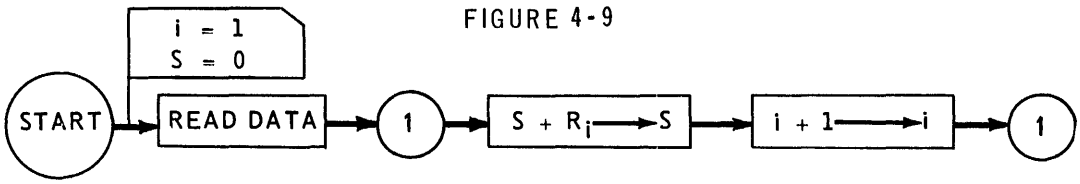


FIGURE 4-8

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

FIGURE 4-9



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 4-5.

	000	READ		read data
			DATA	
1	001	B00880		S + R_i → S
			A00007	
	002	C00007		
			B00001	
	003	L00008		i : 60
			Q00006	
	004	A00009		i + 1 → i
			C00001	
	005	↩		
			U00001	
	006	500007		S → SCP
			900000	stop
	007	↩	↩	S
	008	B00939		constants
			A00007	
	009	000001		
			↩	

STUDENT EXERCISES

1. Reading the data stores 60 quantities of form

$$\pm QQQQQQQQQQ_{\wedge}$$

in cells 880-939. . Print the number of negative quantities.

2. Reading the data stores

   1. a pay of form

$$0_{\bullet}000_{\wedge}00PPPP_{\wedge}PP$$

   in cell 880

   and 2. ten deductions of form

$$0_{\bullet}0000_{\wedge}000DD_{\wedge}DD$$

   in cells 881 - 890.

   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 stores, 60 quantities of form.

$$000000QQQQQQ_{\wedge}$$

in cells 880 - 939. . 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

$$00000BBBB_{\wedge}BB0$$

in cell 880,

2. the employee's shift of form

00000000000S

in cell 881, where S is a key and can take values 1-6,

3. six percentages of form

0PPP00000000

in cells 821 - 826.

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	821
2	822
3	823
4	824
5	825
6	826

Print the pay in form

000000AAAAAA

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.

$$m = 820 + S$$

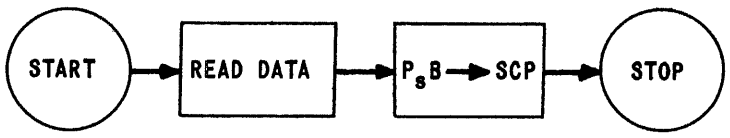
This relationship, or function, can be used to derive the appropriate cell directly from the shift key.

```
000   READ
                DATA        read data
001   B00881
                A00006      Derive cell from shift.
002   C00003
003   L00880
                M0082S      Print pay.
004   C00882
                500882
005   900000                stop
006   L00880
                M00820      constant
```

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 4-10

SHIFT INSTRUCTIONS

n is used to represent a variable second instruction digit.

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

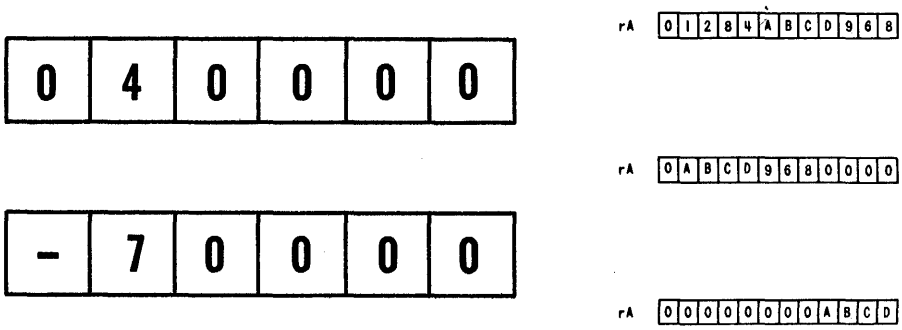


FIGURE 4-11

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.

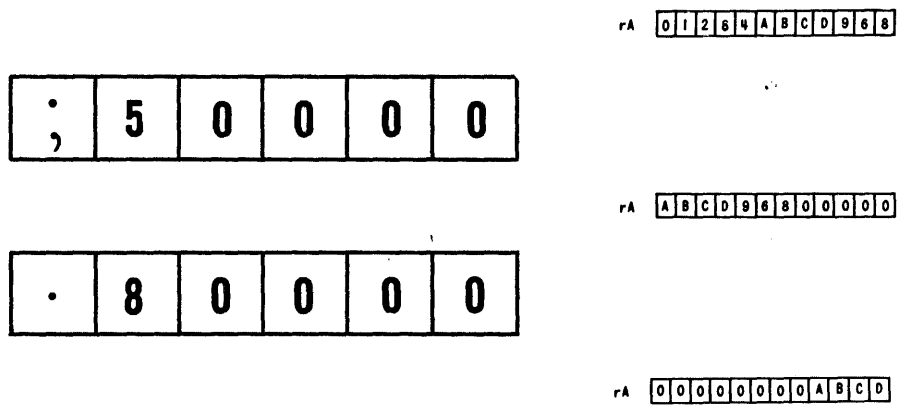


FIGURE 4-12

EXAMPLE

Reading the data stores

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

   00000WWWW000

   in cell 820

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

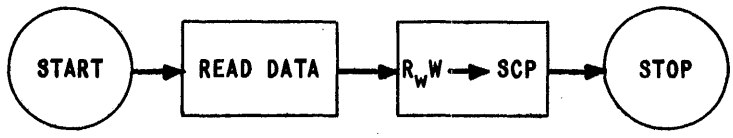
   0RRR00000000

   in cells 900 - 959.

For WEIGHT	apply rate stored in CELL
0000 - 0099	900
0100 - 0199	901
0200 - 0299	902
.	.
.	.
.	.
5900 - 5999	959

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 4-13

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 the address of the cell in which the appropriate rate is stored. Thus, the address of the appropriate cell is 900 added to the two most significant digits of the weight. Dividing the weight by 100 will give the quantity to be added to 900, 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,

$$\frac{4627}{100} = 46.27$$

fractional part
integral part

Thus if W is the weight, the quantity to be added to 900 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.

000	READ	DATA	read data
001	B00820	.50000	
002	A00006	C00003	R_W W → SCP
003	L00820	M009WW	
004	C00821	500821	
005	900000		stop
006	L00820	000000 M00900	constant

EXAMPLE

Reading the data stores

1. the weight in pounds of a package of form

   0000WWWWWW000

   in cell 820

and 2. 60 shipping rates per pound of form

0RRR000000000

in cells 900 - 959

For WEIGHT	apply rate stored in CELL
00000 - 00249	900
00250 - 00499	901
00500 - 00749	902
.	.
.	.
.	.
14750 - 14999	959

Print the cost to ship the package.

FLOW CHART See Figure 4-13

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.

```
000  READ                 }  read data
                DATA
001  L00820               }
                P00006     }
002  A00007               }
                C00003     }  R_W W → SCP
003  L00820               }
                M009WW     }
004  C00821               }
                500821     }
005  900000                 stop
006  000000               }
                .400000    }  constants
007  L00820               }
                M00900     }
```

STUDENT EXERCISES

1. Reading the data stores

DATA	FORM	CELC
Quantity A	±AAAAA^000000	880
Quantity B	±00000BBBBBB^	881

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 880. Print the quantities, each in form

OOOOOOOOQQQQ^

chapter 5

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 59 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: 0OOOOOOO0000
     4: 0RRRRRRR0000
     5: 0PPPPPP000000
```

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 880-939, 6 ten word job items of the form:

```
NNNNNNN00000
0000000CCCCC
0000000LLLLL
0000000MMMMM
0000000OOOOO
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
```

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

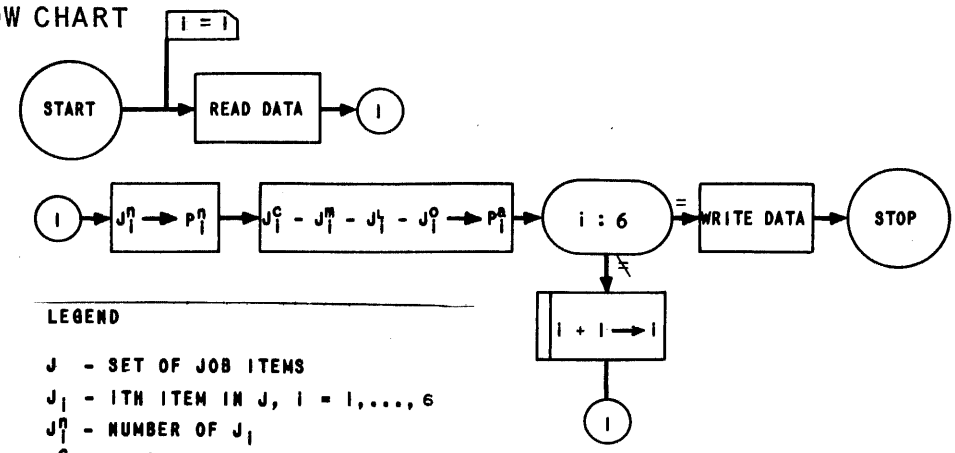
For each job item produce a two word profit item of form:

```
NNNNNNN00000
0000000AAAAA
```

where N - job number
A - profit

Write the profit items.

FLOW CHART



LEGEND

J - SET OF JOB ITEMS
J_i - ITH ITEM IN J, i = 1,...,6
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,...,6
P_i^n - NUMBER OF P_i
P_i^a - PROFIT OF P_i

FIGURE 5-1

CODING

	000	READ		read data
1	001	B00880	DATA C00940	$J_i^N \longrightarrow P_i^N$
	002	B00881	S00882	$J_i^C - J_i^L - J_i^M - J_i^O \longrightarrow P_i^A$
	003	S00883	S00884	
	004	C00941	B00001	
	005	L00014	Q00012	i : 6

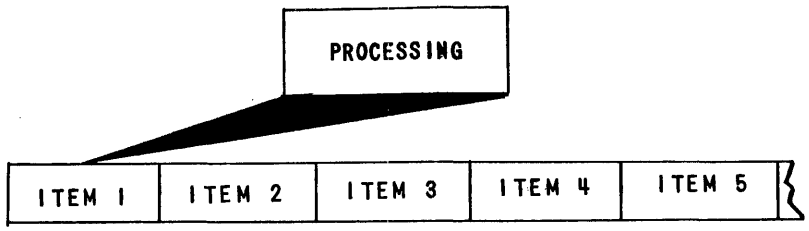
006	A00015		
		C00001	
007	B00002		
		A00016	
008	C00002		
		B00003	
009	A00016		i + 1 ⟶ i
		C00003	
010	B00004		
		A00017	
011	C00004		
		U00001	
012	WRITE		write data
		DATA	
013	900000		stop
		000000	
014	B00930		
		C00950	
015	000010		
		000002	
016	000010		
		000010	
017	000002		
		000000	

WORKING STORAGE

A considerable portion of the above coding is composed of the instructions in cells 004 - 011, 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 5-2



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

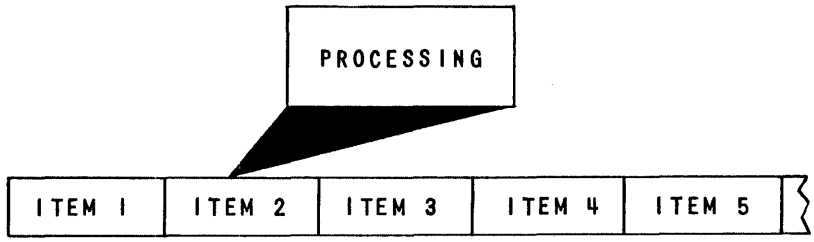


FIGURE 5-3

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 5-2. 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.

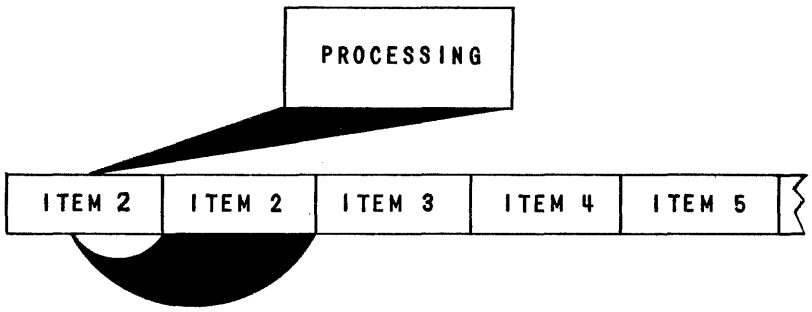


FIGURE 5-4

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.

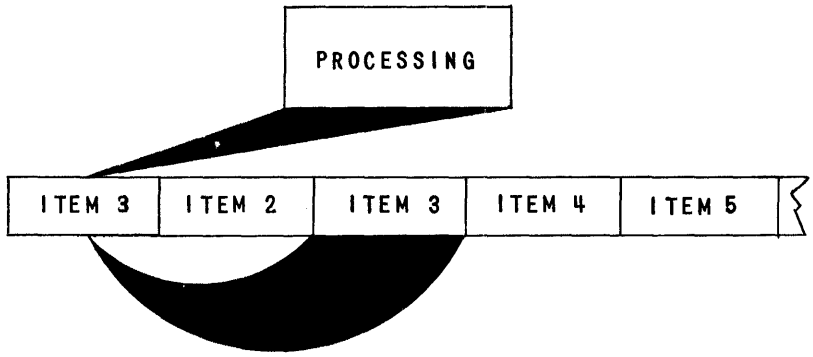


FIGURE 5-5

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 is stored.

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 effect 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 V, rV, a two-word register
Register Y, rY, a ten-word register

Instructions affecting these registers are:

V m, This instruction causes the contents of memory cell m and m + 1 to be duplicated in rV. For our purposes m must be a multiple of two; that is, an address like 000, 102, 504.

W m, This instruction causes the contents of rV to be duplicated in memory cells m and m + 1. Again, m must be a multiple of two.

To illustrate these orders, suppose memory cell 100 contains a quantity A, and cell 101 a quantity B. If the order V 100 is given and at any later time W 304, say, cell 304 contains A and 305 B. The contents of rV are not destroyed upon reading out.

Y m, This instruction causes the contents of cells m, m + 1, ..., m + 9 to be duplicated in rY. m must be a multiple of ten.

Z m, This instruction causes the contents of rY to be duplicated in memory cells m, m + 1, ..., m + 9. Again, m must be a multiple of ten. The contents of rY are not destroyed upon reading out.

For example, if Y 100 is given, then a Z 310,

	100:	A		310:	A
	101:	B		311:	B
And:	102:	C	Then	312:	C
	.	.		.	.
	.	.		.	.
	.	.		.	.
	109:	J		319:	J

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

	000	READ		read data
			DATA	
1	001	B00880		$J_i^N \longrightarrow P_i^N$
			C00950	
	002	B00881		
			S00882	
	003	S00883		
			S00884	$J_i^C - J_i^L - J_i^M - J_i^O \longrightarrow P_i^A$
	004	C00951		
			V00950	
	005	W00940		
			Y00890	
	006	Z00880		
			B00005	i : 6
	007	L00012		
			Q00010	
	008	A00013		i + 1 ⟶ i
			C00005	
	009	000000		
			U00001	
	010	WRITE		write data
			DATA	
	011	900000		stop
			000000	
	012	W00950		
			Y00940	
	013	000002		
			000010	

The item advance coding is in cells 004-009. The variable word is in cell 005. The BOm, LOm and QOm instructions in cells 006 and 007 test i against 6. The variable word will be

W00950Y00940

immediately after the last item has been processed. The Vm instruction in cell 004 and the Wm instruction in cell 005 transfer the output item just produced from output working storage to its proper location in the output area. The Ym instruction in cell 005 and the Zm instruction in cell 006 transfer the next input item from its location in the input area to input working storage. The AOm and COm instructions in cell 008 increase the addresses of the instructions in the variable word. The UOm instruction in cell 009 transfers control to the processing instructions.

STUDENT EXERCISE

Reading the data stores, in cells 880-939, 6 ten word inventory items of form

0000000NNNNN
0000000HHHHH
0000000OOOOO
0000000RRRRR
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX

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

and in cells 820-831, 6 two word sales items of form

0000000NNNNN
0000000QQQQQ

where N - stock number
Q - sales quantity

The inventory item in cells 880-889 and the sales item in cells 820 and 821 have the same stock number; the item in cells 890-899 and the item in cells 822 and 823 have the same stock number; the items in cells 900-909 and the item in cells 824 and 825 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)$\longrightarrow$rF	Fill

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

INSTRUCTION	OPERATION
GOm	(rF)$\longrightarrow$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 1-18.

INSTRUCTION	OPERATION	MNEMONIC
EOm	"odd" characters of (rF) extract (m)$\longrightarrow$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.

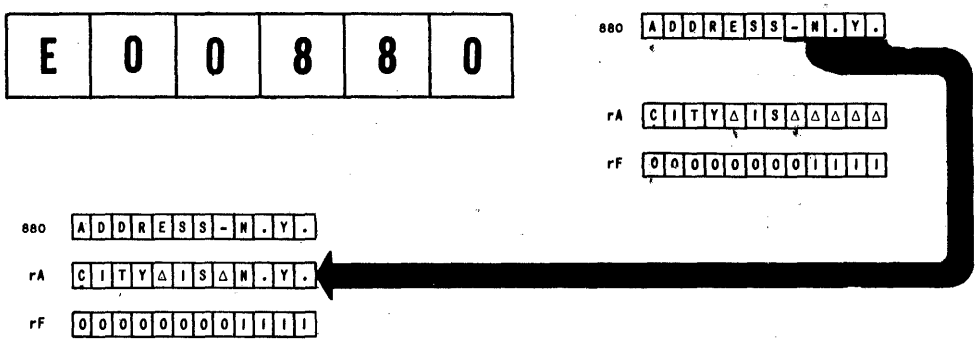


FIGURE 5-6

ILLUSTRATIVE EXAMPLE

In memory cell 100 is a quantity in the following format:

000 XXX XXX XXX

which we desire to print on the supervisory control printer, suppressing the non-significant zeros. That is, if 100: 000 000 690 760 we wish to print 690 760 only.

000	L00100		000 XXX XXX XXX ⟶ rL
		B00006	001 - - - - - - - - - - - ⟶ rA
001	– 10000		Shift rA right one place
		T00001	Transfer control if rA > rL
002	H00008		rA ⟶ rF
		F00008	
003	B00007		
		E00100	ΔΔΔ ΔΔΔ ΔΔΔ ΔΔΔ ⟶ rA
004	H00008		
		500008	Print edited quantity
005	900000		Stop;
		000000	
006	001- - -		
		- - - - - -	
007	ΔΔΔΔΔΔ		
		ΔΔΔΔΔΔ	
008	–		
		–	Working Storage

Note that in line 001, the iterative portion of the routine, the word in rA is successively shifted to the right until the 1 lines up under the left-most non-zero digit of the word in cell 100. The ones and dashes then can be used as an extractor, replacing the space symbols with the most significant digit of 100 and all digits to its right. The space symbols, of course, move the typewriter carriage but do not print. If we were to print a column of numbers edited in this fashion, they would be aligned on the least significant digit. If we wished them aligned on the most significant digit, we would replace line 007 with ignores: iii iii iii iii.

STUDENT EXERCISES

1. Reading the data stores, in cell 880 and 881, two one word items of form

0AAA0BBB0CCC

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

00000000SSSS

2. Reading the data stores

DATA	FORM	CELL
Quantity A	000AAAA00000	880
Quantity B	000000BBBBBB	881

Print the sum of the quantities in form

000SSSSSSSSS

3. Reading the data stores in cells 880- 939, 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. 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

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

3. Transaction Item

FIELD	NUMBER OF CHARACTERS
Key	8
Transaction Code	4
Transaction Information	12

chapter 6

Subroutines and Variable Connectors

COMMON SUBROUTINES

ILLUSTRATIVE EXAMPLE

Reading the data stores, in cells 880-939, 6 ten word job items of form

```
SSSSSSSSSSS
0000000PPPPP
0000000LLLLL
0000000MMMMM
0000000OOOOO
XXXXXXXXXXX
XXXXXXXXXXX
XXXXXXXXXXX
XXXXXXXXXXX
XXXXXXXXXXX
```

where S - salesman code and can be

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

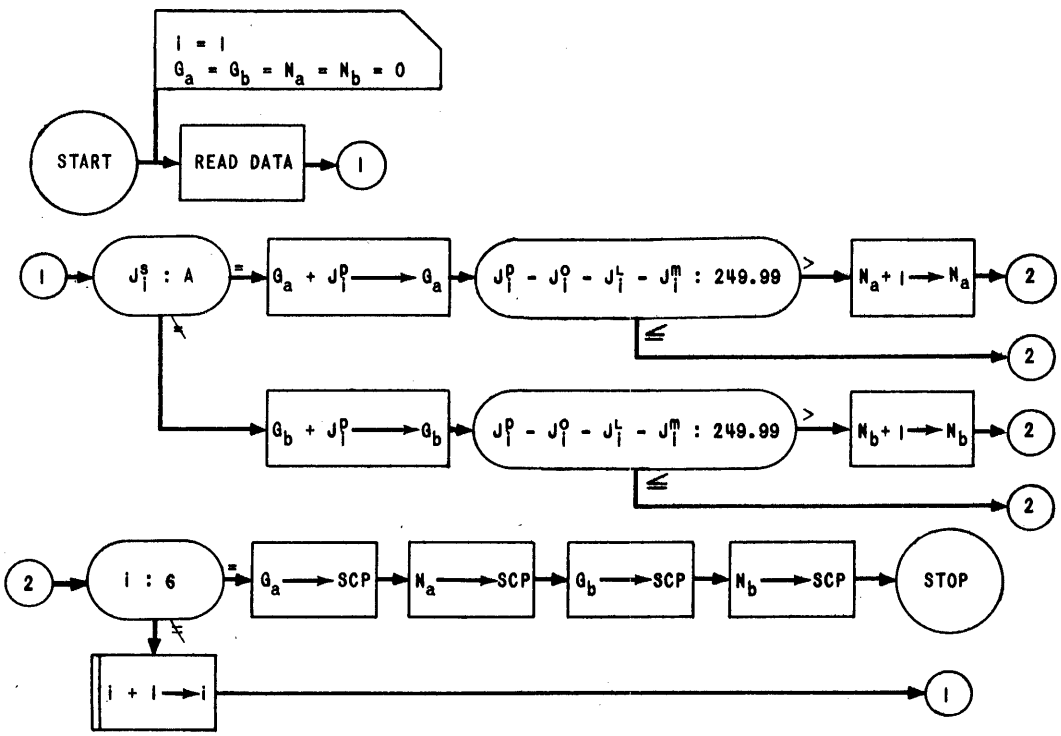
P - contract price
L - labor cost
M - material cost
O - overhead cost
X - other data

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,..., 6
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 6-1

	000	READ		
			DATA	read data
1	001	B00880		
			L00030	$J_i^S : A$
	002	000000		
			Q00013	
	003	B00027		
			A00881	$G_B + J_i^P \longrightarrow G_B$
	004	C00027		
			B00881	
	005	300882		
			S00883	
	006	S00884		$J_i^P - J_i^O - J_i^L - J_i^M : 249.99$
			L00031	
	007	000000		
			T00021	
2	008	Y00890		
			Z00880	
	009	B00008		
			L00032	$i : 6$
	010	000000		
			Q00023	
	011	A00033		
			C00008	$i + 1 \longrightarrow i$
	012	000000		
			U00001	
	013	B00026		
			A00881	
	014	C00026		$G_A + J_i^P \longrightarrow G_A$
			B00881	
	015	S00882		
			S00883	
	016	S00884		$J_i^P - J_i^O - J_i^L - J_i^M : 249.99$
			L00031	
	017	000000		
			T00019	
	018	000000		
			U00008	
	019	B00028		
			A00034	$N_A + 1 \longrightarrow N_A$
	020	C00028		
			U00008	
	021	B00029		
			A00034	$N_B + 1 \longrightarrow N_B$
	022	C00029		
			U00008	
	023	500026		$G_A \longrightarrow SCP ; N_A \longrightarrow SCP$
			500028	
	024	500027		$G_B \longrightarrow SCP ; N_B \longrightarrow SCP$
			500029	

025	900000		stop
		00 000	
026	000000		G_A
		000000	
027	000000		G_B
		000000	
028	000000		N_A
		000000	
029	000000		N_B
		000000	
030	AAAAAA		
		AAAAAA	
031	000000		
		024999	
032	Y00940		
		Z00880	
033	000010		
		000000	
034	000000		
		000001	

The coding in cells 004-007 is duplicated in cells 014-017. 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.

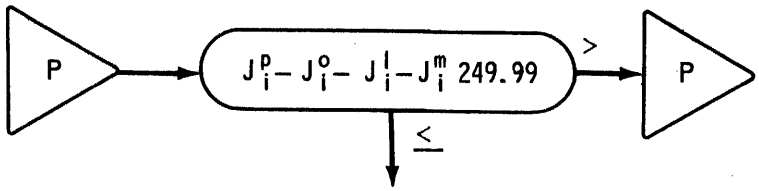
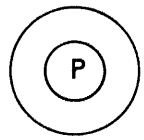


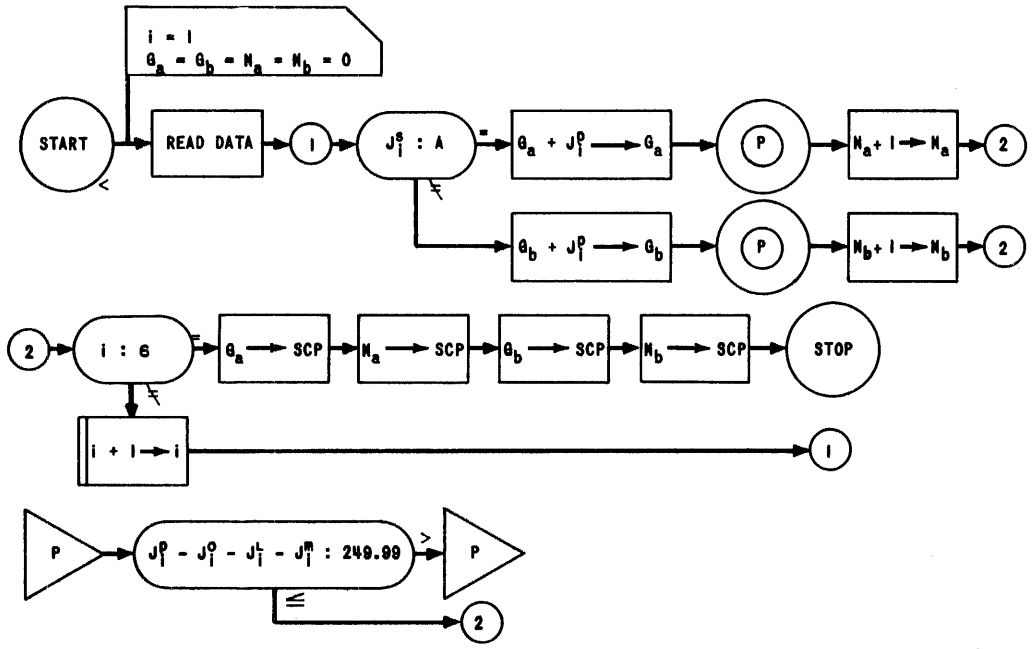
FIGURE 6-2

For example, in the following flow chart (Figure 6-4) after the operation, $G_A + J_i^p \longrightarrow G_A$, the subroutine symbol

FIGURE 6-3



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,..., 6
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 6-4

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 005 and also by means of a UOm instruction in cell 010, 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, two zeros and the three least significant digits of (CC) in m, or record 000000U0(CC) in m.

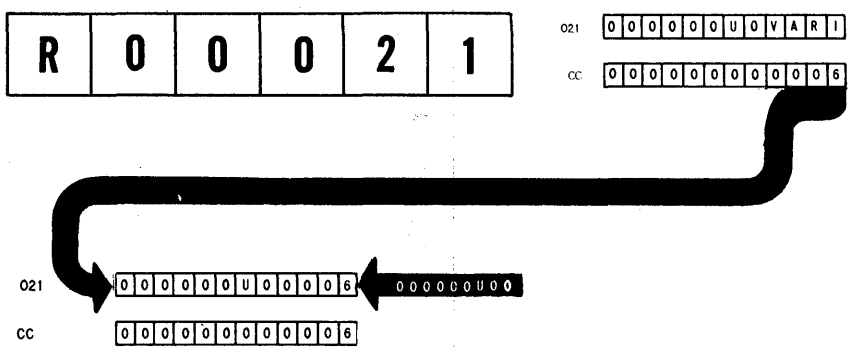


FIGURE 6-5

Consider the following. (alpha time has just been completed) 000000U00

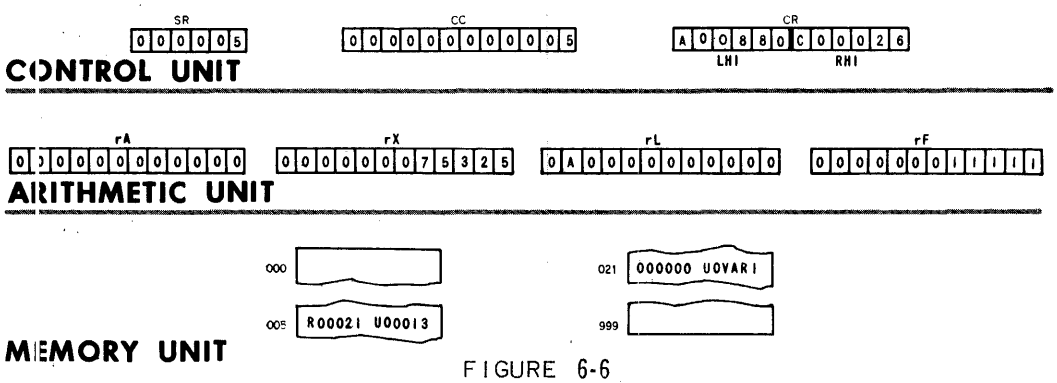


FIGURE 6-6

On beta time

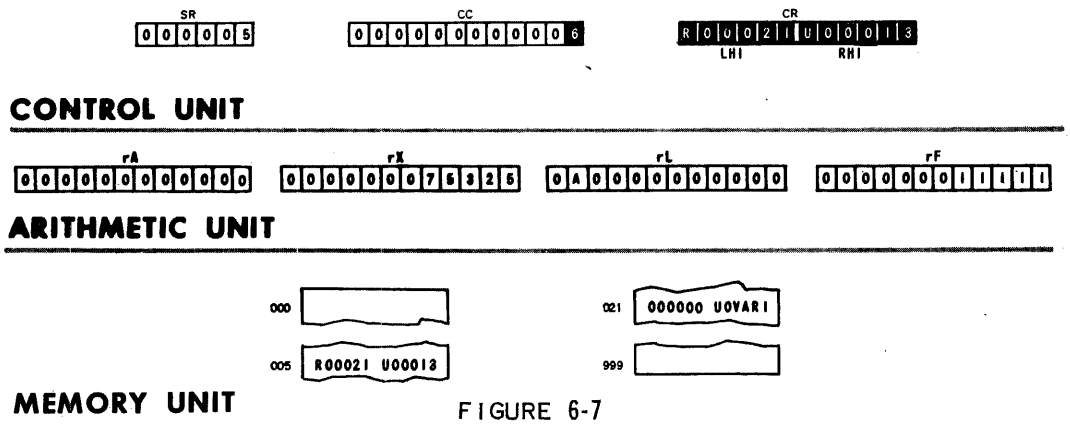


FIGURE 6-7

On gamma time

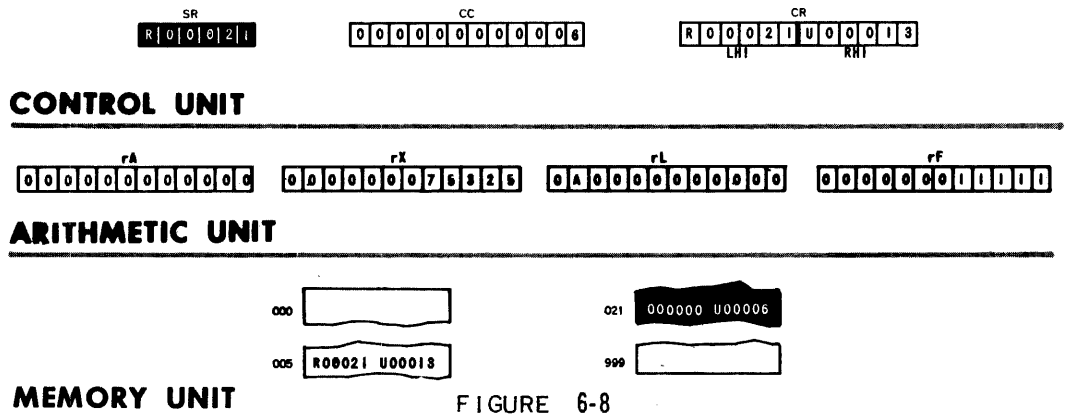


FIGURE 6-8

On delta time

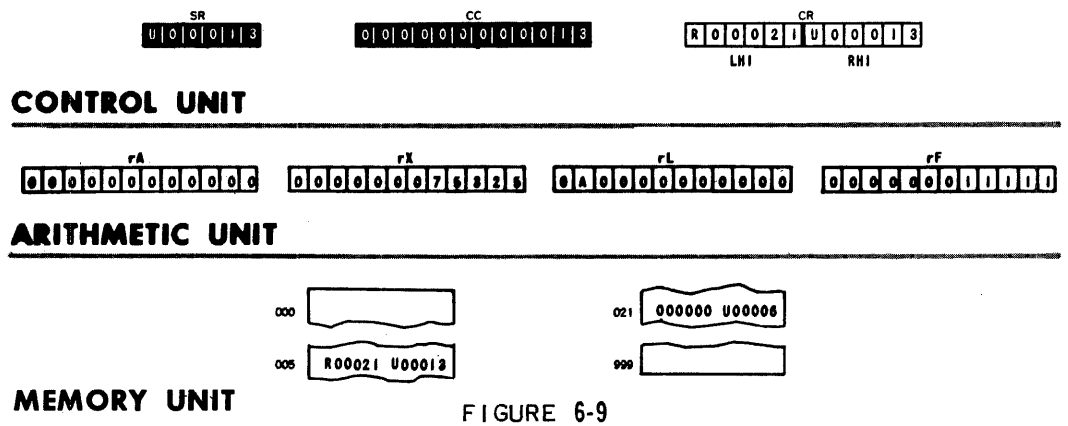


FIGURE 6-9

If cell 013 were the entrance of a common subroutine; and cell 021, the exit; and if the common subroutine were to be entered from cell 005; the execution of the

UCm 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

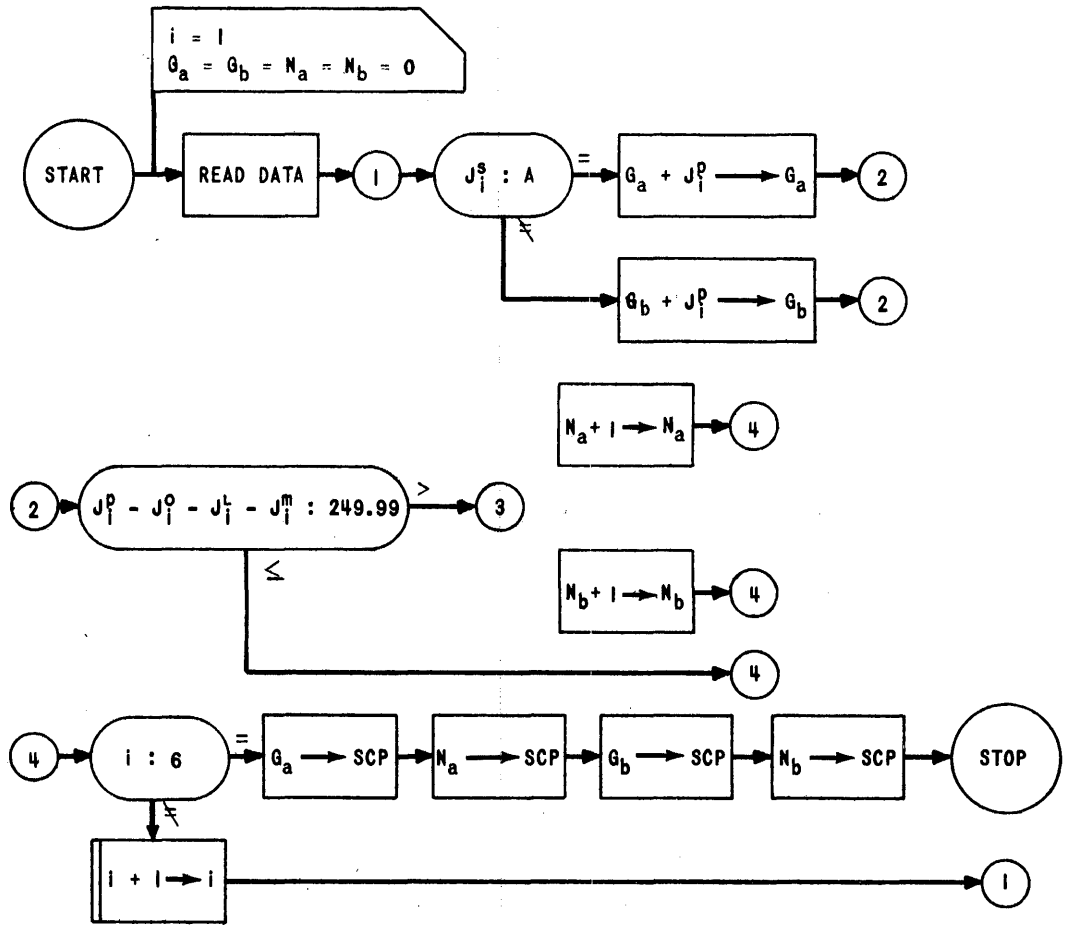
will be executed, transferring control to cell 006, to continue the processing begun before transferring to the subroutine.

	000	READ		
			DATA	read data
1	001	B00880		
			L00029	$J_i^S : A$
	002	000000		
			Q00008	
	003	B00026		
			A00881	$G_B + J_i^P \longrightarrow 6_B$
	004	C00026		
			000000	
	005	R00021		P
			U00013	
	006	B00028		
			A00030	$N_B + 1 \longrightarrow N_B$
	007	C00028		
			U00016	
	008	B00025		
			A00881	$G_A + J_i^P \longrightarrow G_A$
	009	C00025		
			000000	
	010	R00021		P
			U00013	
	011	B00027		
			A00030	$N_A + 1 \longrightarrow N_A$
	012	C00027		
			U00016	
P	013	B00881		
			S00882	
	014	S00883		$J_i^P - J_i^O - J_i^L - J_i^M : 249.99$
			S00884	
	015	L00031		
			T00021	
2	016	Y00890		
			Z00880	
	017	B00016		
			L00032	$i : 6$
	018	000000		
			Q00022	
	019	A00033		
			C00016	$i + 1 \longrightarrow i$
	020	000000		
			U00001	

021	000000	U00VAR	P
022	500025	500027	$G_A \rightarrow SCP$; $6_A \rightarrow SCP$
023	500026	500028	$G_B \rightarrow SCP$; $6_B \rightarrow SCP$
024	900000	000000	stop
025	000000	000000	G_A
026	000000	000000	G_B
027	000000	000000	N_A
028	000000	000000	N_B
029	AAAAAA	AAAAAA	
030	000000	000001	
031	000000	024999	
032	Y00940	Z00880	
033	000010	000000	

VARIABLE CONNECTORS

The example can be flow charted in another way. (The notation from 3 to 4 in figure 6-10 is incomplete).



LEGEND

J - A SET OF JOB ITEMS
J_i - THE ITH ITEM IN J, i = 1,..., 6
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 6-10

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.

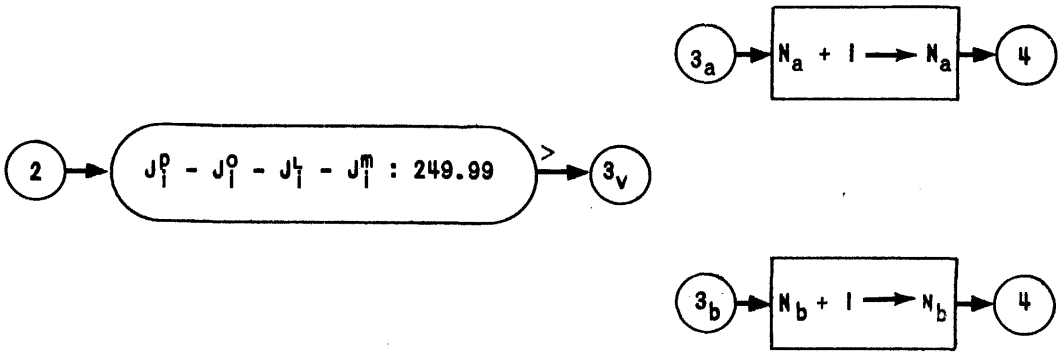


FIGURE 6-11

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

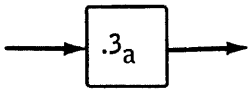
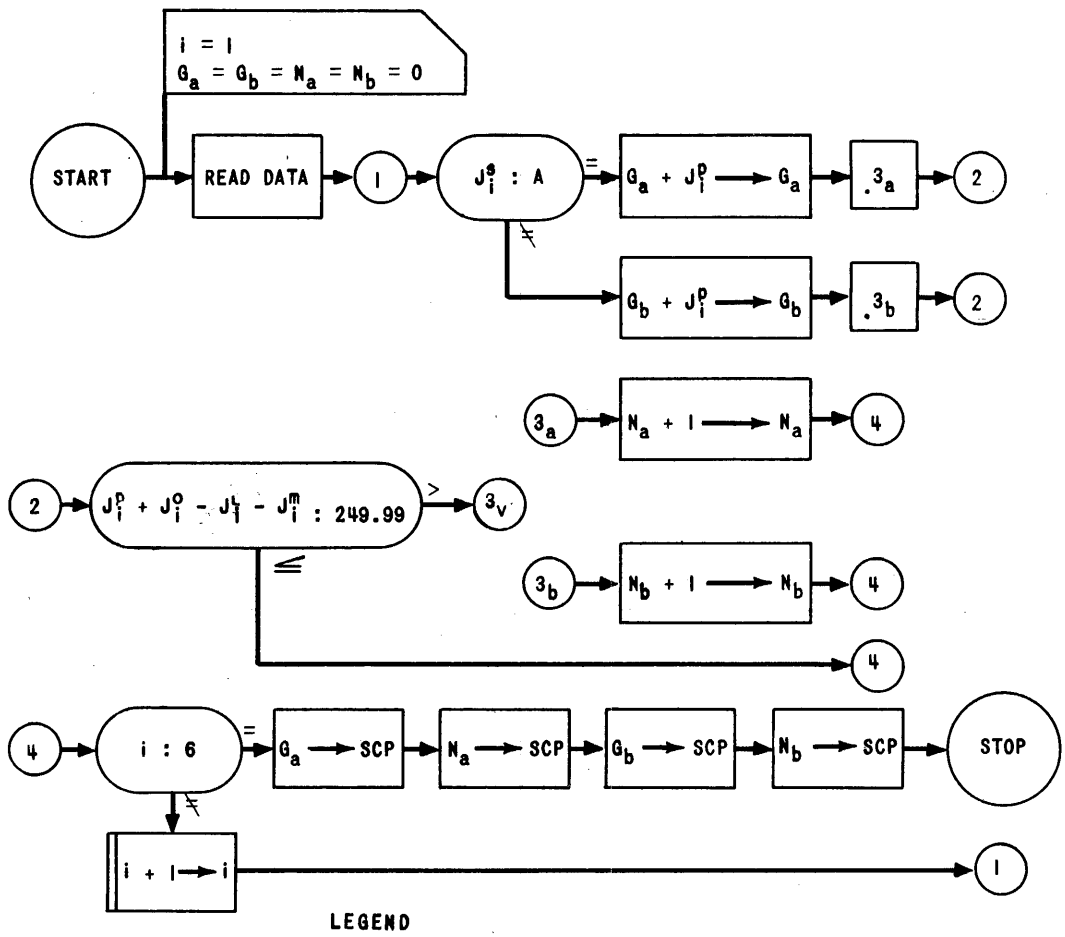


FIGURE 6-12

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.



LEGEND

J - SET OF JOB ITEMS
J_i - ITH ITEM IN J, i = 1,..., 6
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 6-13

This flow chart is logically equivalent to the flow chart in figure 6-4 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.

	000	READ		
			DATA	
1	001	B00880		read data
			L00027	J_i^S : A
	002	000000		
			Q00006	
	003	B00024		
			A00881	
	004	C00024		$G_B + J_i^P \longrightarrow 6_B$
			B00028	
	005	C00011		
			U00009	.3b
	006	B00023		
			A00881	$G_A + J_i^P \longrightarrow G_A$
	007	C00023		
			B00029	
	008	C00011		
			000000	.3a
2	009	B00881		
			S00882	
	010	S00883		$J_i^P - J_i^O - J_i^L - J_i^M$: 249.99
			S00883	
3r	011	L00030		
			T00VAR	
4	012	Y00890		
			Z00880	
	013	B00012		
			L00031	i : 6
	014	000000		
			Q00021	
	015	A00032		
			C00012	$i + 1 \longrightarrow i$
	016	000000		
			U00001	
3a	017	B00025		
			A00033	$N_A + 1 \longrightarrow N_A$
	018	C00025		
			U00012	
3b	019	B00026		
			A00033	$N_B + 1 \longrightarrow N_B$
	020	C00026		
			U00012	

021	500023		$G_A \longrightarrow SCP$; $N_A \longrightarrow SCP$
		500025	
022	500024		$G_B \longrightarrow SCP$; $N_B \longrightarrow SCP$
		500026	
023	000000		G_A
		000000	
024	000000		G_B
		000000	
025	000000		N_A
		000000	
026	000000		N_B
		000000	
027	AAAAAA		
		AAAAAA	
028	L00030		
		T00019	
029	L00030		
		T00017	
030	000000		
		024999	
031	Y00940		
		Z00880	
032	000010		
		000000	
033	000000		
		000001	

In this coding variable connector three is embodied in the address part of the TOm instruction in cell 011. This address part varies between 017 and 019, 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 cells 007 and 008 to pole b by the pair in cells 004 and 005.

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 880-939
2. six ten word B items in cells 820-879

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.

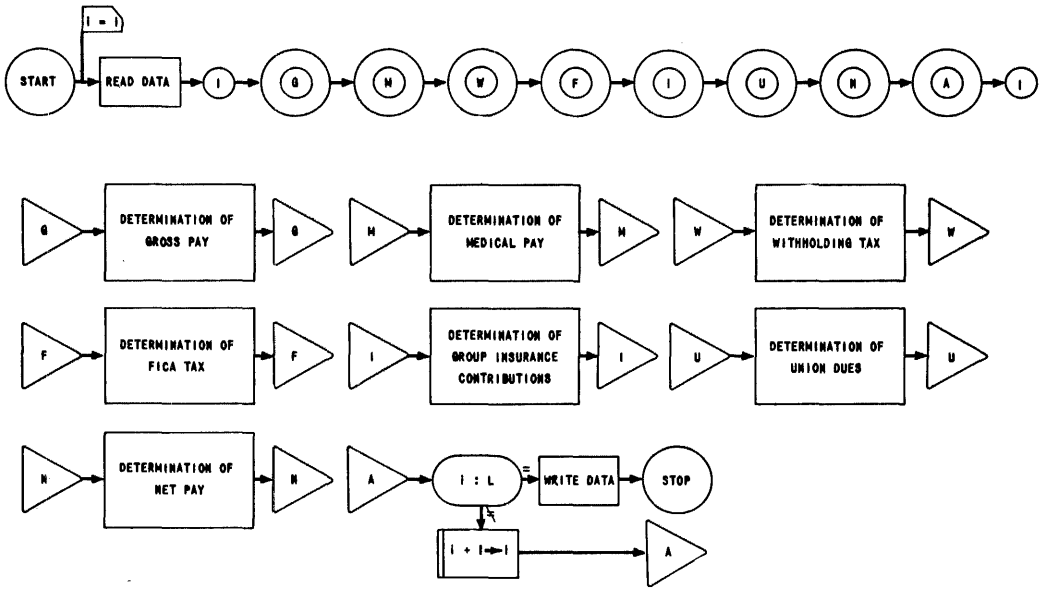


FIGURE 6-14

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.

chapter 7

Detailed Description of Instructions

TRANSFER OF CONTROL INSTRUCTIONS

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 three 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 820 are greater than or equal to the contents of cell 880, and another is to be taken if the contents of cell 820 are less than the contents of cell 880. The coding might be

```
010   B00820
                 L00880
011   ⌣
                 T00020
012   ⌣
                 Q00020
```

In this coding, the LHI in cells 011 and 012 are wasted, since they are skips. It would be more efficient if the QOm instruction were the LHI in cell 011. This situation is possible, since the address part of the QOm will still be the three least significant digits of the word in which it appears.

```
010   B00820
                 L00880
011   Q00000
                 T00020
```

If the contents of cells 820 and 880 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.

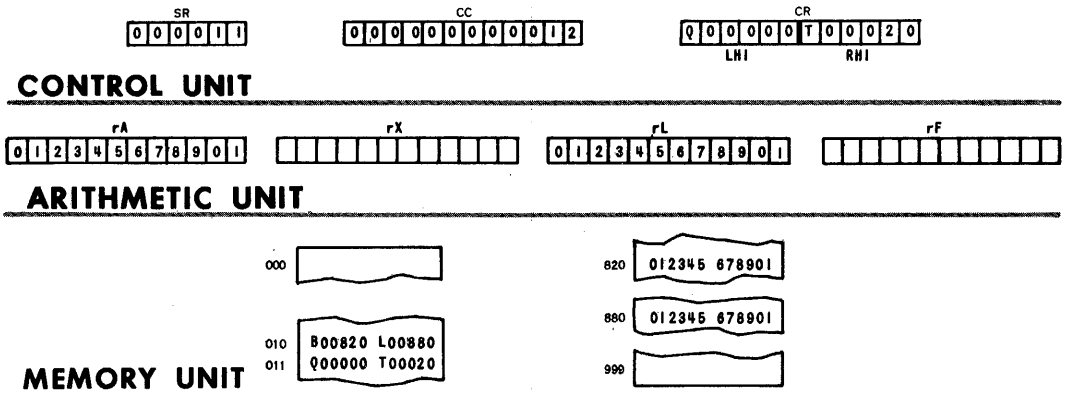


FIGURE 7-1

Assume that the computer has just completed beta time. (CC) specify that the next instruction pair is in cell 012. On gamma time Q00000 is transferred to SR and executed. Since (rA) are equal to (rL), the execution transfers the three least significant digits of (CR) to CC.

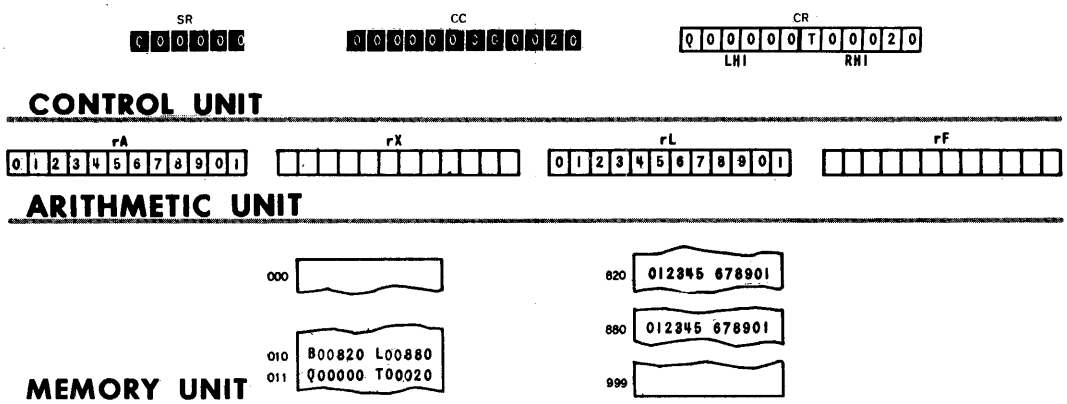


FIGURE 7-2

(CC) now specify that the next instruction pair is in cell 020, where the coding for the condition of equality begins. On delta time T00020 is transferred to SR and executed. Since (rA) are not greater than (rL), T00020 is interpreted as a skip.

SHIFT INSTRUCTIONS

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

REGISTER V

If the m in both the Vm and Wm instructions is odd and the least significant digit is not equal to nine, the instructions behave as in the following example.

Example: (051) = a, (052) = b. Transfer "a" and "b" to 063 and 064 respectively.

MEMORY LOCATION	INSTRUCTION	REMARKS
020	V00051	a, b ⟶ rV
	W00063	a ⟶ 063; b ⟶ 064;
		(rV) = a, b.

If the m in one instruction is odd (least significant digit not equal to nine), and the m of the other instruction is even, the two words are transferred in reversed order.

Example: (051) = a, (052) = b. Transfer "b" followed by "a" to 054 and 055 respectively.

MEMORY LOCATION	INSTRUCTION	REMARKS
020	V00051	a,b ⟶ rV
	W00054	b ⟶ 054; a ⟶ 055;
		(rV) = a,b

Example: (050) = a, (051) = b. Transfer "b" followed by "a" to 063 and 064 respectively.

MEMORY LOCATION	INSTRUCTION	REMARKS
020	V00050	a, b ⟶ rV
	W00063	b ⟶ 063; a ⟶ 064; (rV) = a, b

If the m in a Vm or Wm instruction has a nine as its least significant digit, the instruction will transfer from, or to, the last and first words in the ten-word memory channel.

Example: (050) = a, (059) = b. Transfer "a" followed by "b" to 100 and 101.

MEMORY LOCATION	INSTRUCTION	REMARKS
020	V00059	b, a ⟶ rV
	W00100	a ⟶ 100; b ⟶ 101; (rV) = b, a.
	REGISTER Y	

When executing a Ym or Zm instruction, the least significant digit of m is ignored by the computer. The transfers operate on the integral multiples of ten. Thus, Y999 is equivalent to Y990, and Z784 to Z780.

ARITHMETIC INSTRUCTIONS

ADD INSTRUCTIONS

Some details of the add instructions have been given on page 59 . 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.

$$\begin{array}{r} - \\ +\ 6 \\ \hline 5 \end{array} \quad \begin{array}{r} - \\ +\ - \\ \hline \Delta \end{array} \quad \begin{array}{r} \& \\ +\ 5 \\ \hline 16 \end{array}$$

SUBTRACT INSTRUCTIONS

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 1-18 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 880 contains

B12345678901

and the instruction

S00880

is executed, rX will contain

A12345678901

MULTIPLY INSTRUCTIONS

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

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

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 7-3

THE DIVIDE INSTRUCTION

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 1-18, 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 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. 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, B00880 would be written as B 880, but still recorded as B00880.

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, if a negative number of eleven significant integers is subtracted from a posi-

tive number of eleven significant integers there can be overflow.

$$\begin{array}{r} +.50000000000 \\ -(-.50000000000) \\ \hline 1.00000000000 \end{array}$$

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 gamma or delta time. The carry into the sign position is lost. If overflow occurs on gamma time, delta time will be executed. At the end of the cycle during which overflow occured, the following special four stage cycle is executed.

Alpha Time - six zeros are transferred to the static register.

Beta Time - the contents of memory cell 000 are transferred to the control register; one is <u>not</u> added to the contents of the control counter.

Gamma Time - the left hand instruction of the contents of the control register is transferred to the static register and executed.

Delta Time - the right hand instruction is transferred to the static register and executed.

On the succeeding four stage cycle, control returns to the pair of instructions in the memory cell specified by the contents of the control counter. The contents of the control counter were one greater than the address of the memory cell containing the instruction being executed when overflow occurred. If overflow occurred due to an instruction in memory cell k, then the instructions in memory cell k+1, now specified by the present contents of the control counter, will be executed, provided that neither memory cell k, nor memory cell 000, 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. Memory cell 000 must contain some sort of instruction, usually a transfer of control instruction, to assure that the instructions for taking the new course of action will be executed when overflow occurs.

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. 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. The variable word which contains the counter and the variable address might initially have the following appearance,

V70882W00880

and the following constant could be added to it.

001002000000

In summary, overflow permits an alternate course of action based on the decision, "have all the items in the set been processed?" The instruction pair stored in memory cell 000 can be used to transfer control to, or execute, the routine which is to be performed on reaching this limit. 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 (stored in memory cells 880-939) is exhausted, stop the computer.

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

```
[020   V00882
                W00880]
 021   B00020
                L00026
 022   000000
                Q00025   }   i : 30
 023   A00027
                C00020      i + 1 ---> i
 024   000000
                U00XXX      to processing
 025   900000
                000000      stop
 026   V00940
                W00880
 027   000002
                000000
```

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.

```
[000   B00023
                C00000]
   .
   .            }   processing
   .
[020   V70882
                W00880]
 021   B00020
                A00024 *    i + 1 ---> i
 022   C00020
                U00XXX      to processing
 023   900000   000000
 024   001002
                000000
```

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

The routine operates as follows. Initially the B0m C0m instruction pair in memory cell 000 provides a method for storing the necessary stop instruction in memory cell 000 without actually having to execute the stop instruction. When control initially reaches the item advance routine, the contents of memory cells 882 and 883 are transferred to memory cells 880 and 881. The contents of memory cell 020 and the constant for advancing the address and the counter are added, and the sum is transferred to memory cell 020. As a result, memory cell 020 now contains

V71884W00880

Control is then transferred to processing. After processing 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 memory cell 020 are successively

V72886W00880
V73888W00880
V74890W00880

and so on until the thirtieth item is processed. At that point the contents of cell 020 are

V99940W00880

After processing the 30th item control returns to the item advance subroutine. The contents of memory cells 940 and 941 are transferred to memory cells 880 and 881. The contents of memory cell 020 are transferred to register A. The execution of the A0m instruction adds one to the 99 in the second and third digit positions of the contents of register A, and overflow occurs. The carry is lost. On alpha time six zeros are transferred to the static register. On beta time the contents of memory cell 000

900000000000

are transferred to the control register. On gamma time the left-hand instruction of the contents of the control register

900000

is transferred to the static register and executed, thus stopping the computer.

The above is an example of "specialized overflow". The overflow is called specialized, because no matter in what memory cell overflow occurs, the result is some specific operation, namely, the computer stops.

If overflow is used to control more than one iterative process in the same routine and if the course of action to be taken when one of the iterative processes reaches its limit is different from the course of action to be taken when another of the iterative processes reaches its limit, it is obvious that specialized overflow will not be able to handle the situation. In such a case, "generalized overflow" is necessary. Consider the following.

ILLUSTRATIVE EXAMPLE:

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

0AAAAAAAAAAA

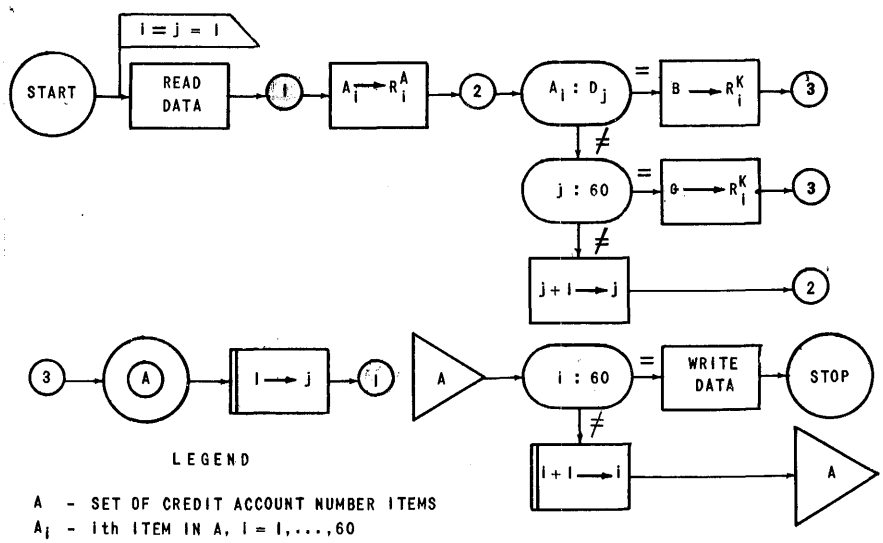
in cells 820-879, and 60 one word delinquent account number items of form

0DDDDDDDDDDD

in cells 880-939. 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.



LEGEND

A - SET OF CREDIT ACCOUNT NUMBER ITEMS
A_i - ith ITEM IN A, $i = 1,\ldots,60$
D - SET OF DELINQUENT ACCOUNT NUMBER ITEMS
D_j - jth ITEM IN D, $j = 1,\ldots,60$
R - SET OF CREDIT ITEMS
R_i - ith ITEM IN R
R_i^A - ACCOUNT NUMBER OF R_i
R_i^K - KEY OF R_i

FIGURE 7-4

CODING

It has already been demonstrated that overflow can be used to control an item advance subroutine. The flow chart in figure 7-4 indicates that the process to be followed when the delinquent account number item advance reaches its limit is not the same as the process to be followed when the credit item advance reaches its limit. Consequently, if overflow is to be used to control both item advances, generalized overflow must be used. The following coding incorporates generalized overflow.

```
000  R00004
                U00002
001  000000
                000001
002  B00004
                A00001
003  C00004
                000000
004  000000
                U00VAR
005  READ                          read data
                DATA
(2)(1) 006  B00820
                L00880            A_i : D_j
007  000000
                Q00017
008  B40881
                C00880
009  B00008
                000000
010  A00026                        j + 1 ---> j
                C00008*
011  000000
                U00006
012  B00820
                F00027
013  E00028                        g ---> R_i^K
                C00999
(3) 014  R00023                    (A)
                U00019
015  B00029                        1 ---> j
                C00008
016  000000
                U00006
017  F00027
                E00030
018  C00999                        B ---> R_i^K
(A)             U00014
019  B00999
                000000
020  C40940
                B00821
021  C00820
                B00020
022  A00031                        i + 1 ---> i
                C00020*
023  000000                        (A)
                U00015
```

024	WRITE		write data
		DATA	
025	900000		stop
		000000	
026	001001		
		000000	
027	100000		
		000000	
028	G00000		
		000000	
029	B40881		
		C00880	
030	B00000		
		000000	
031	001001		
		000001	

The coding for the generalized overflow subroutine appears in memory cells 000-004 of this coding. To see how the generalized overflow subroutine works, consider one of the item advance subroutines that uses it, for example, the credit item advance, which appears in memory cells 019-023. This item advance operates in the same manner as the one used to demonstrate specialized overflow. When overflow occurs the contents of the control counter are

000000000023

During the special four stage cycle that results from the overflow the following occurs. On alpha time six zeros are transferred to the static register. On beta time the contents of memory cell 000

R00004U00002

are transferred to the control register. The execution of the R0m instruction transfers the word.

000000U00023

to memory cell 004. The execution of the U0m instruction transfers control to cell 002. The constant

000000000001

is added to the contents of memory cell 004, and the sum

000000U00024

is transferred to memory cell 004. On the next four stage cycle the 00mU0m instruction pair just fabricated is executed, thus transferring control to memory cell 0024, where the coding for the process to be followed when the credit item advances reaches its limit begins.

A closer inspection of the credit item advance subroutine will reveal the following structure. If the instruction causing overflow is considered to be stored in memory cell k (memory cell 022 in the subroutine being considered), then the contents of memory cell k+1 relate to the normal item advance subroutine and are not executed when overflow occurs, and memory cell k+2 contains the coding for the beginning of the process to be followed when the item advance reaches its limit. Investigation will reveal that the delinquent account number item advance subroutine stored in memory cells 008-011 embodies the same structure. As a matter of fact, this structure is general for any subroutine taking advantage of the generalized overflow subroutine shown in memory cells 000-004. The only caution that must be observed in the use of generalized overflow is that, no matter where or how many times in a routine overflow is used for control purposes, the subroutine to be followed when overflow occurs must be coded two memory cells below the memory cell containing the instruction on 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

   000000QQQQQQ
   ^

   in cells 880-939.

   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 six ten word A items in cells 820-879 and six ten word B items in cells 880-939.. 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.

chapter 8

Input – Output

Magnetic tape is the means of introducing, and removing, large volumes of data to, and from, the memory. The tape is metal about one half inch wide and .002 inches thick. Data may be written on a tape, read, 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 being about 1550 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.

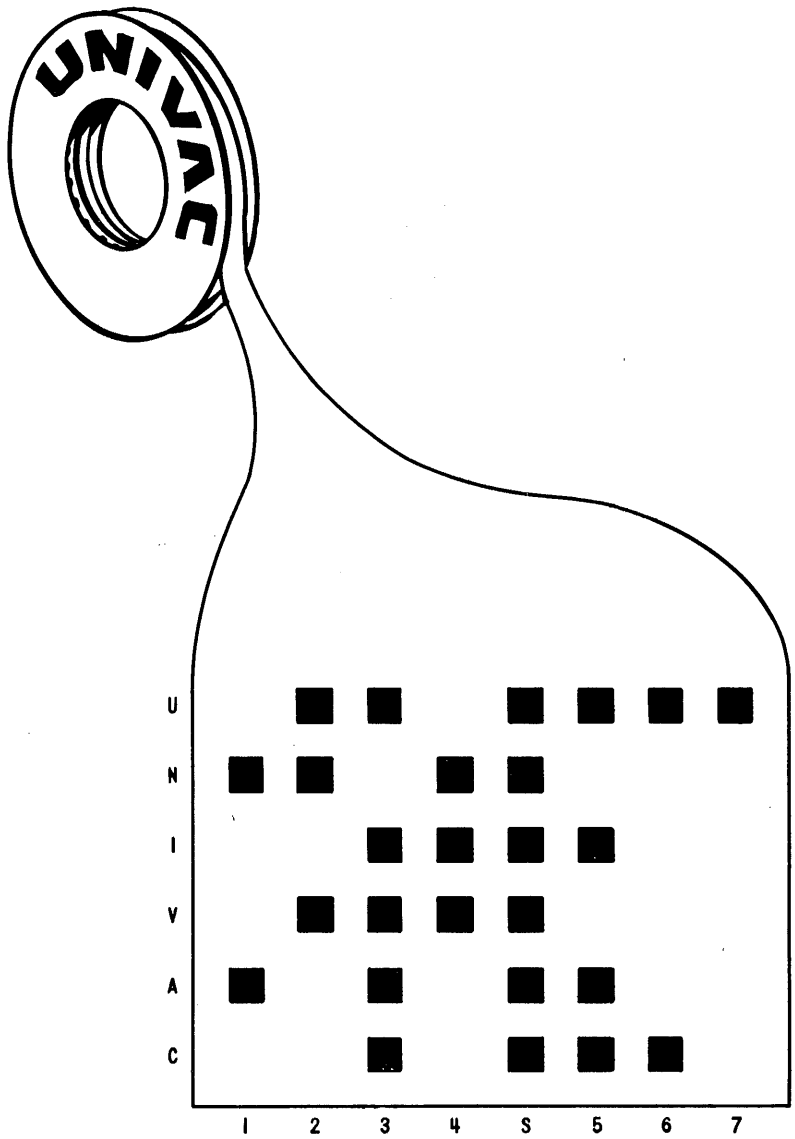


FIGURE 8-1

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.

CODE COMBINATIONS OF
THE 63 UNIVAC I CHARACTERS

FIGURE 8-2

	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

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 —.

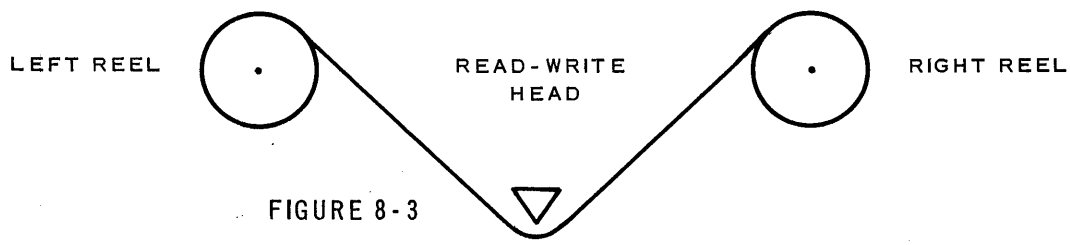


FIGURE 8-3

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 8-4. 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 8-4.

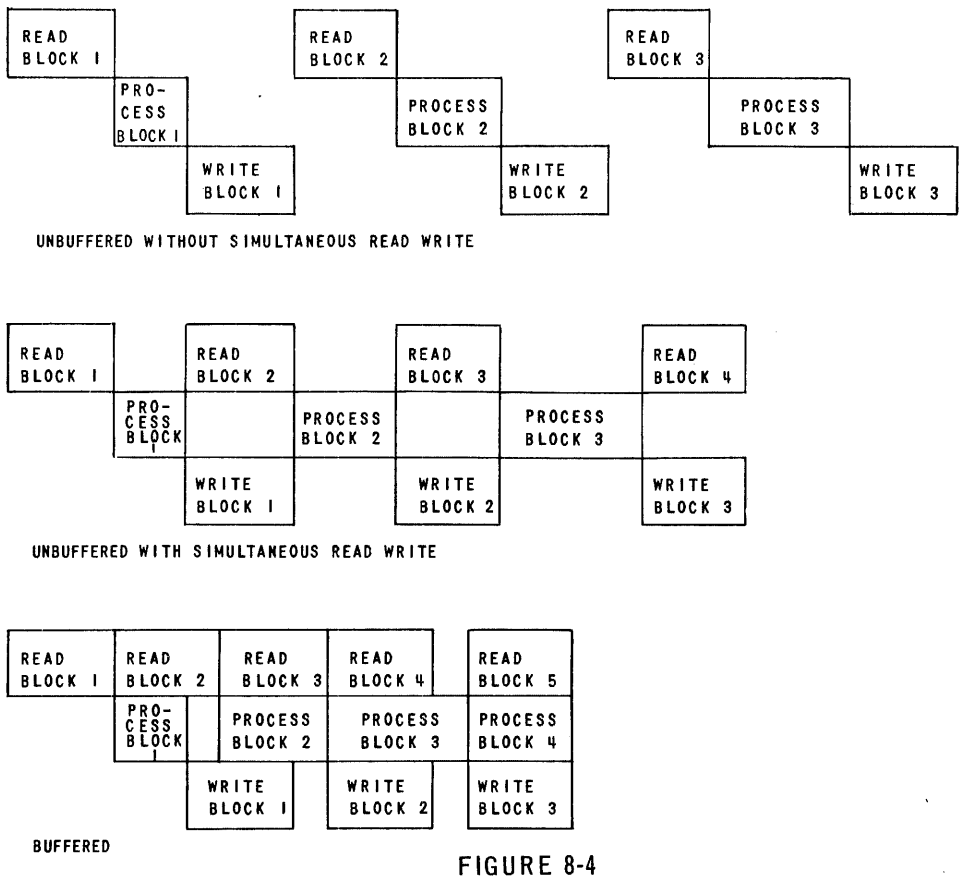


FIGURE 8-4

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

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.

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

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

NOTE: m must be a location ending in zero.(This applies to all tape instructions.) 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 128 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 20 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.

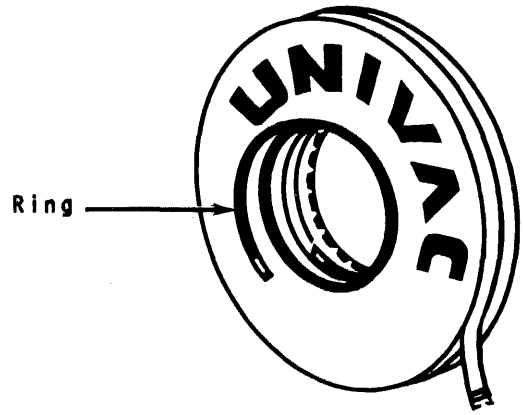


FIGURE 8-5

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, completing 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 zero word 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 zero and the last words of the block.

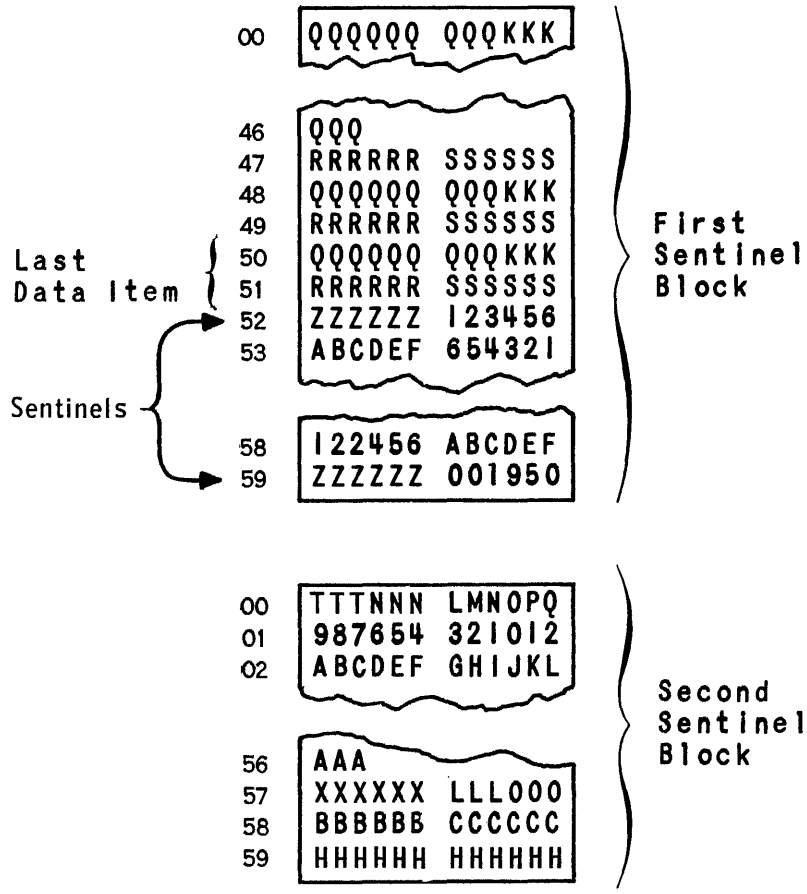


FIGURE 8-6

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 000-059. 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 10 buttons called Initial 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 Initial Tape Selector button is depressed.

ILLUSTRATIVE EXAMPLE

A tape contains a series of ten word job items of form

```
NNNNNNNNNNNN
0000000CCCCC
0000000LLLLL
0000000MMMMM
0000000OOOOO
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
XXXXXXXXXXXX
```

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

There is at least one full block of data on the tape.

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

```
NNNNNNNNNNNN
0000000PPPPP
```

where N - job number
P - profit

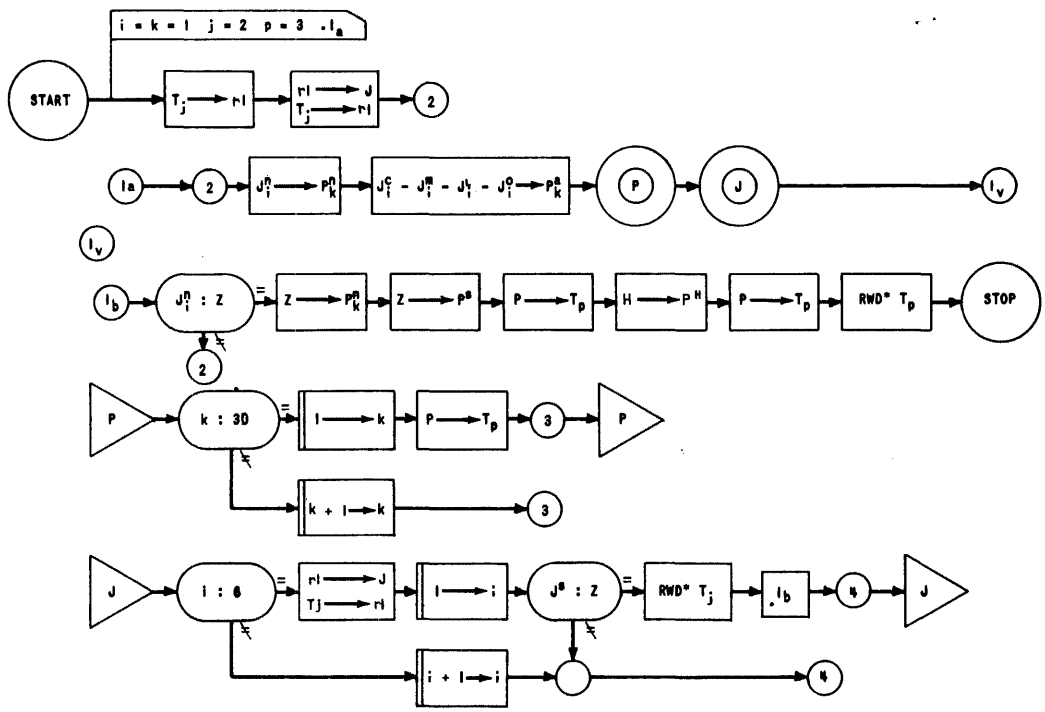
Write the profit items.

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,..., 6
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,..., 30
P_k^n - NUMBER OF P_k
P_k^a - AMOUNT OF P_k
P^s - SENTINEL OF P
P^H LAST WORD OF 2ND SENTINEL BLOCK

FIGURE 8-7

The following is a description of the thinking that might have accompanied this flow chart.

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.

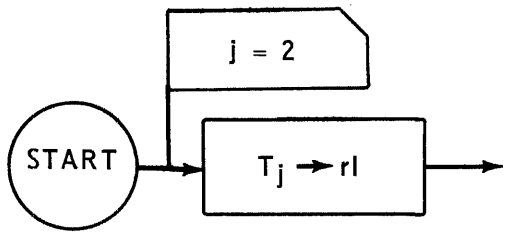


FIGURE 8-8

(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.

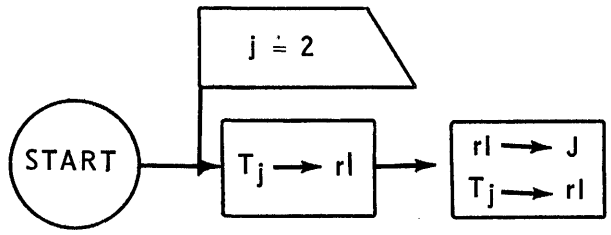


FIGURE 8-9

With a block of job items in the memory processing can begin.

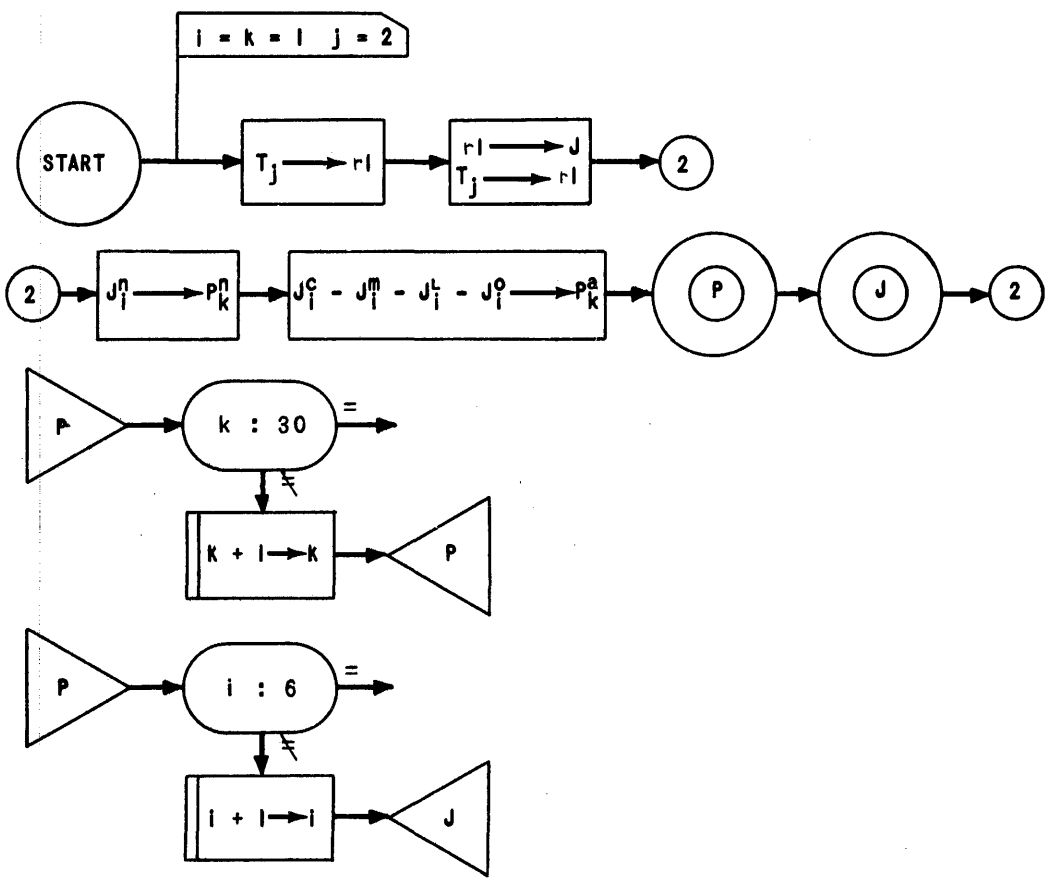


FIGURE 8-10

When a block of job items is exhausted the input item counter equals 6. 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.

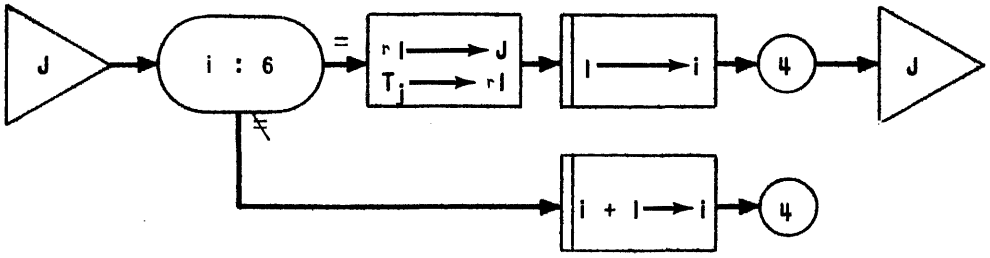


FIGURE 8-11

When the output block is filled, the output item counter will equal 30. The output item counter is reset to one to prepare for the next output block, and the current output block is written.

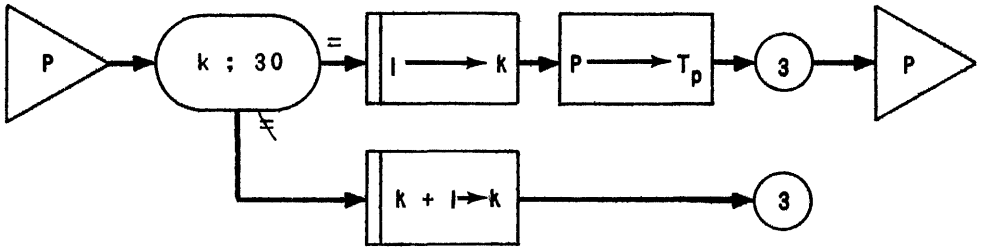


FIGURE 8-12

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.

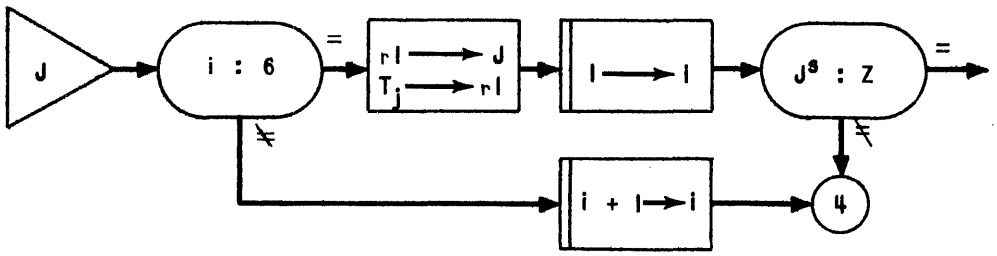


FIGURE 8-13

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.

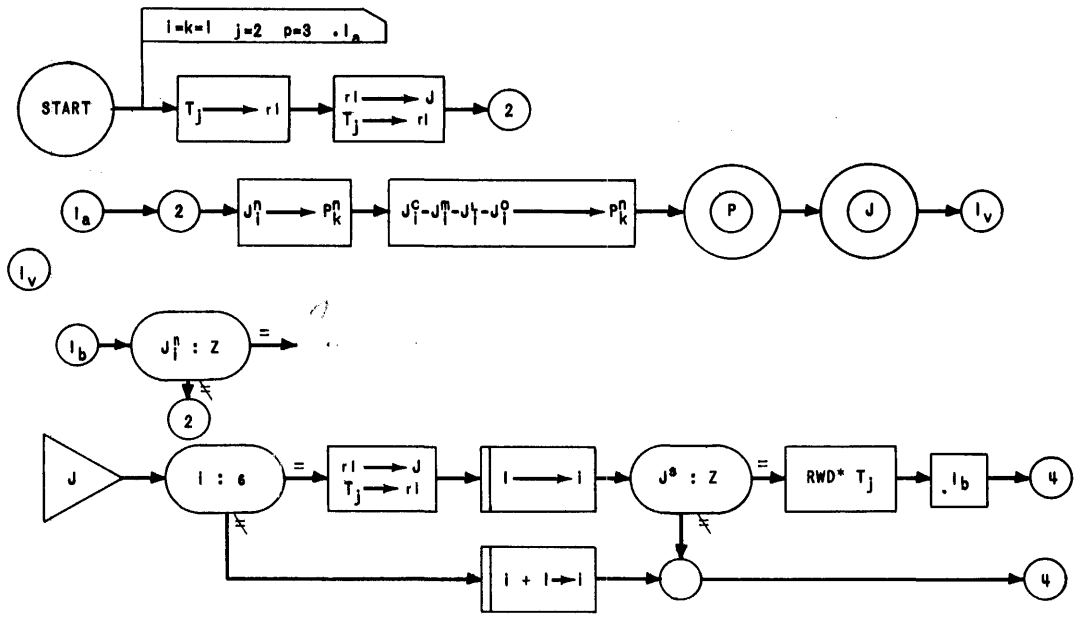


FIGURE 8-14

When the key of the item to be processed is a sentinel all the data has been processed. Sentinels must be written on T_p. The last block of output is in the memory. A sentinel must be stored in the zero word 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 stored 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. Words 00 through 58 of the block currently in the output area constitute the "hash" desired. A constant consisting of HHHHHHHHHHHH is placed in word 59 to insure that the last four digits of this word will not contain numerics. This block is then 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
940-999	Output
880-939	Input

	000	R00004	U00002	
	001	000000	000001	
	002	B00004	A00001	
	003	C00004	000000	
	004	000000	U00VAR	
	005	B00007	L00010	
	006	000000	Q00012	
	007	110000	300060	
	008	A00011	C00007	
	009	000000	U00005	
	010	110000	300060	
	011	000000	000060	
	012	810000	120000	$T_j \longrightarrow rI$
	013	320880	000000	$rI \longrightarrow J;\ T_j \longrightarrow rI$
1a 1v	014	B00880	C00998	$J_i^N \longrightarrow P_i^N$

2	015	B00881	S00882	$J_i^C - J_i^M - J_i^L - J_i^O \longrightarrow P_k^A$
	016	S00883	S00884	
	017	C00999	000000	
	018	R00024	U00021	
	019	R00030	U00027	
	020	000000	U00014	
P	021	V70998	W00940	
	022	B00021	000000	
	023	A00048	C00021 *	k : 30
3	024	000000	U00019	
	025	B00049	C00021	$1 \longrightarrow k$
	026	530940	U00024	$P \longrightarrow T_p$
J	027	Y40890	Z00880	
	028	B00027	000000	
	029	A00050	C00027 *	$i + 1 \longrightarrow i$
4	030	000000	U00020	
	031	320880	B00051	$rI \longrightarrow J$; $T_j \longrightarrow rI$
	032	C00027	F00052	$1 \longrightarrow i$
	033	E00939	L00053	
	034	000000	Q00036	$J^S : Z$
	035	000000	U00030	
	036	820000	000000	RWD * T_j
	037	R00014	U00030	.1b
1b	038	K00000	F00052	
	039	E00880	L00053	$J_i^N : Z$
	040	000000	Q00043	

041	B00880		$J_i^N \longrightarrow P_k^N$
		C00998	
042	000000		
		U00015	
043	H00998		$Z \longrightarrow P^S$
		H00999	
044	R00022		$Z \longrightarrow P_k^N$
		U00021	
045	530940		$P \longrightarrow T_p$; $H \longrightarrow P^H$
		B00054	
046	C00999		$P \longrightarrow T_p$;
		530940	
047	830000		RWD * T_p ; Stop
		900000	
048	001000		
		000002	
049	V70998		
		W00940	
050	010010		
		000000	
051	Y40890		
		Z00880	
052	111111		
		000000	
053	ZZZZZZ		
		000000	
054	HHHHHH		
		HHHHHH	

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 025, 031 and 032.

To store a sentinel in the zero word 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 WOm instruction in cell 021. The sentinel is transferred to 998 by the HOm instruction in cell 043. The UOm instruction in cell 044 transfers control to the VmWm instruction pair, which transfers the sentinel. The ROm instruction in cell 044 guarantees that, after the WOm instruction has been executed, control returns to cell 045 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^
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^
```

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.

chapter 9

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
0QQQQQ000000
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 sales items of form

```
NNNNNNNNNNNN
0AAAAA000000
```

Where N - stock number
A - quantity

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.

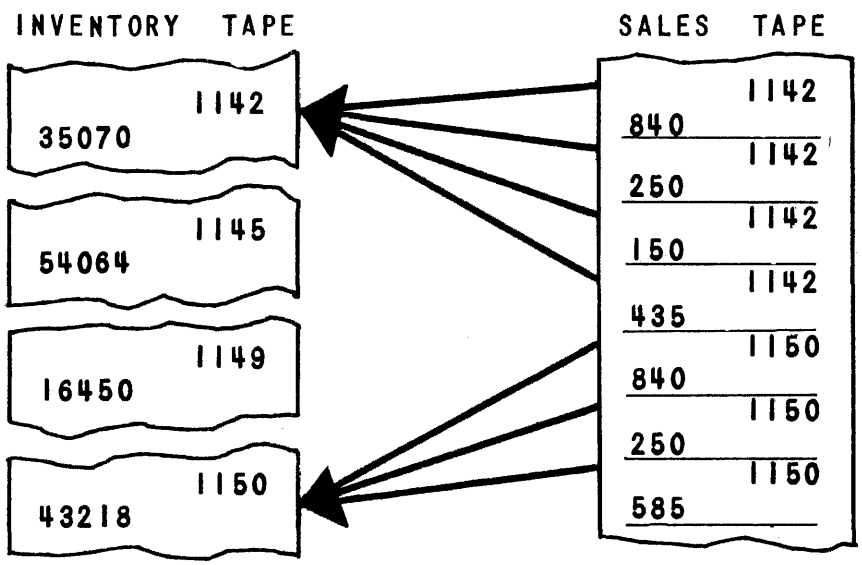


FIGURE 9-1

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.

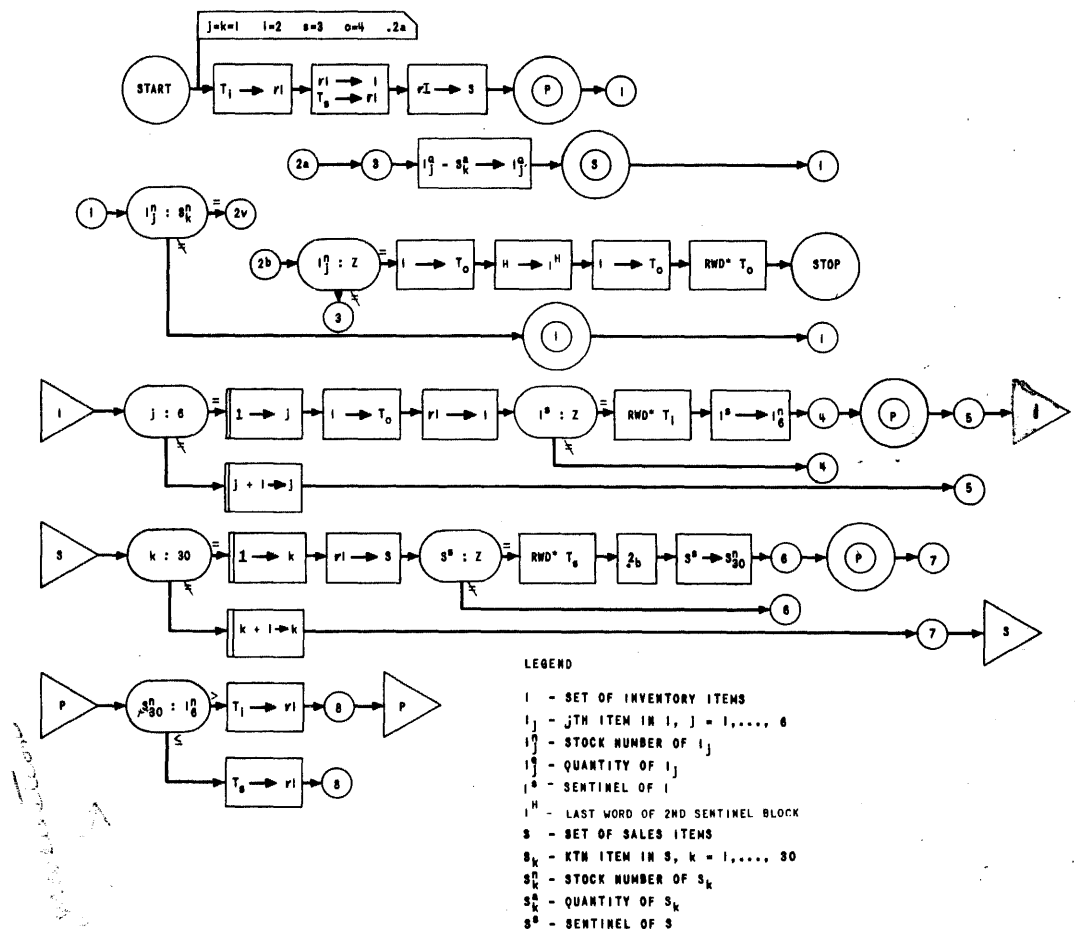


FIGURE 9-2

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

820 - 879 - Sales Area
880 - 939 - Inventory Input Area
940 - 999 - Inventory Output Area

CODING

```
000   R00004
                U00002
001   000000
                000001
002   B00004
                A00001
003   C00004
                000000
004   000000
                U00VAR
005   B00007
                L00010
006   000000
                Q00012
007   110000
                300060
008   A00011
                C00007
009   000000
                U00005
010   110000
                300120
011   000000
                000060
012   810000
                120000
013   330880
                300820
014   R00056
                U00052
015   B00880
                L00820
016   000000
                Q00019
017   R00027
                U00023
```

1

$T_i \longrightarrow rI$

$rI \longrightarrow I$; $T_S \longrightarrow rI$; $rI \longrightarrow S$

P

$I_j^N : S_k^N$

I

	018	000000		
			U00015	
2a 2v	019	B00881		
			000000	
3	020	S00821		$I_j^Q - S_k^A \longrightarrow I_j^Q$
			C00881	
	021	R00039		
			U00036	S
	022	000000		
			U00015	
I	023	Y00880		
			B00024	
	024	Z40940		
			Y00890	
	025	Z00880		$j + 1 \longrightarrow j$
			000000	
	026	A00062	*	
			C00024	
5	027	000000		I
			U00018	
	028	B00063		$1 \longrightarrow j$
			C00024	
	029	540940		$I \longrightarrow T_O$; $rI \longrightarrow I$
			300880	
	030	F00064		
			E00939	$I^S : Z$
	031	L00065		
			Q00034	
4	032	R00056		P
			U00052	
	033	000000		
			U00027	RWD * T_i ; $I^S \longrightarrow I_6^N$
	034	820000		
			C00930	
	035	000000		
S			U00032	
	036	V70822		
			W00820	
	037	B00036		
			000000	$k + 1 \longrightarrow k$
	038	A00066		
7			C00036 *	S
	039	000000		
			U00022	$1 \longrightarrow k$
	040	B00067		
			C00036	$rI \longrightarrow S$
	041	300820		
			F00064	
	042	E00789		
			L00065	$S^S : Z$
	043	000000		
			Q00046	

6	044	R00056		
			U00052	
	045	000000		
			U00039	
	046	830000		RWD * T_S ; $S^S \longrightarrow S^N_{30}$
			C00878	
	047	R00019		.2b
			U00044	
2b	048	K00000		
			F00064	
	049	E00880		
			L00065	I^N_j : Z
	050	000000		
			Q00057	
	051	B00881		
			U00020	
P	052	B00878		
			L00930	
	053	000000		S^N_{30} : I^N_6
			T00055	
	054	130000		$T_S \longrightarrow rI$
			U00056	
	055	120000		$T_i \longrightarrow rI$
			000000	
8	056	000000		
			U00VAR	
	057	R00025		
			U00023	
	058	B00065		
			H00999	
	059	540940		$I \longrightarrow T_O$; $Z \longrightarrow I^N_1$
			B00068	
	060	C00999		$I \longrightarrow T_O$; RWD * T_O
			540940	
	061	840000		stop
			900000	
	062	010010		
			000010	
	063	Z40940		
			Y00890	
	064	111111		
			000000	
	065	ZZZZZZ		
			000000	
	066	001002		
			000000	
	067	V70822		
			W00820	
	068	HHHHHH		
			HHHHHH	

P

STUDENT EXERCISE

A tape contains a series of ten 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

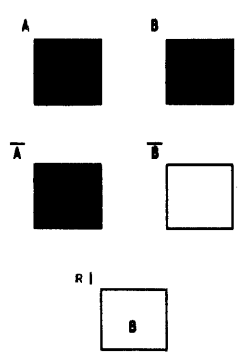
No two policy number items are the same. The items are in ascending order by policy number on both tapes. There is at least one full block of data on each tape. Write a tape containing the policy items for which there is a policy number item on the policy number tape.

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.



CONFIGURATION I
FIGURE 9-3

The following discussion of the operation of the standby block technique is based on figure 9-4.

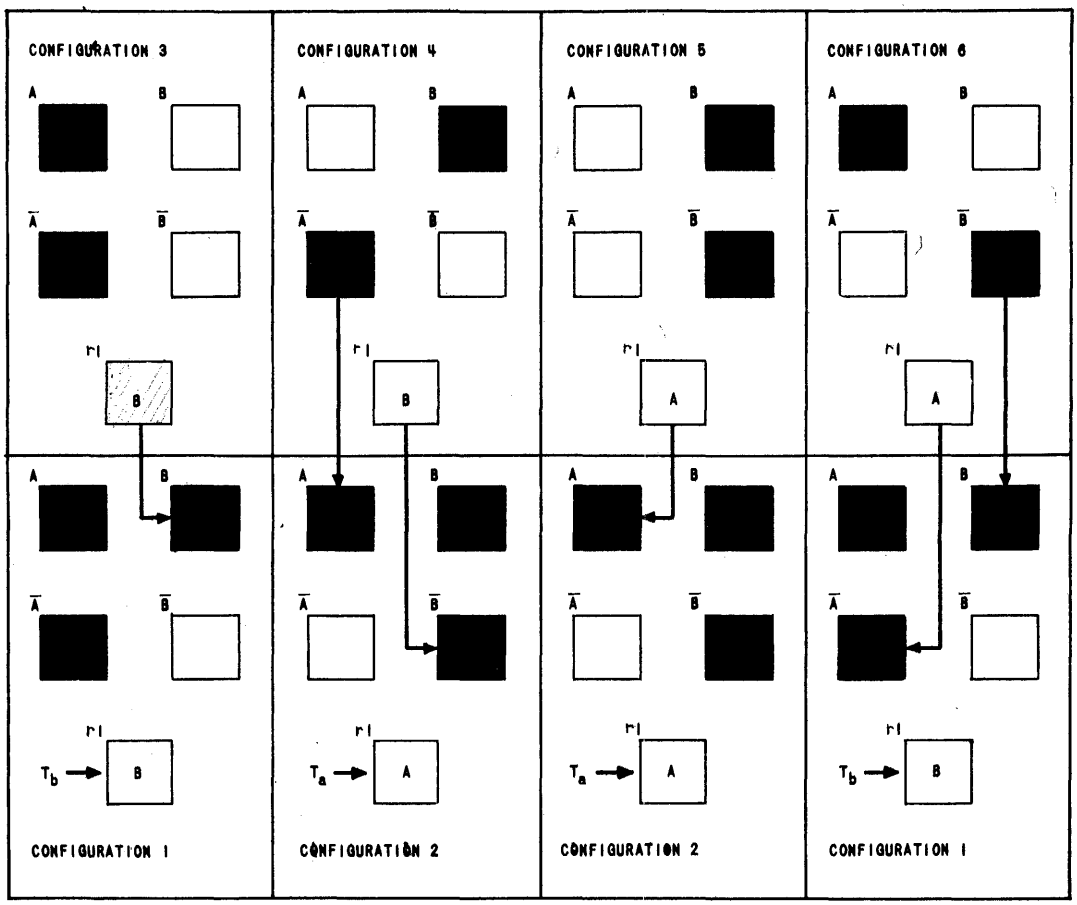


FIGURE 9-4

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 1.

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.

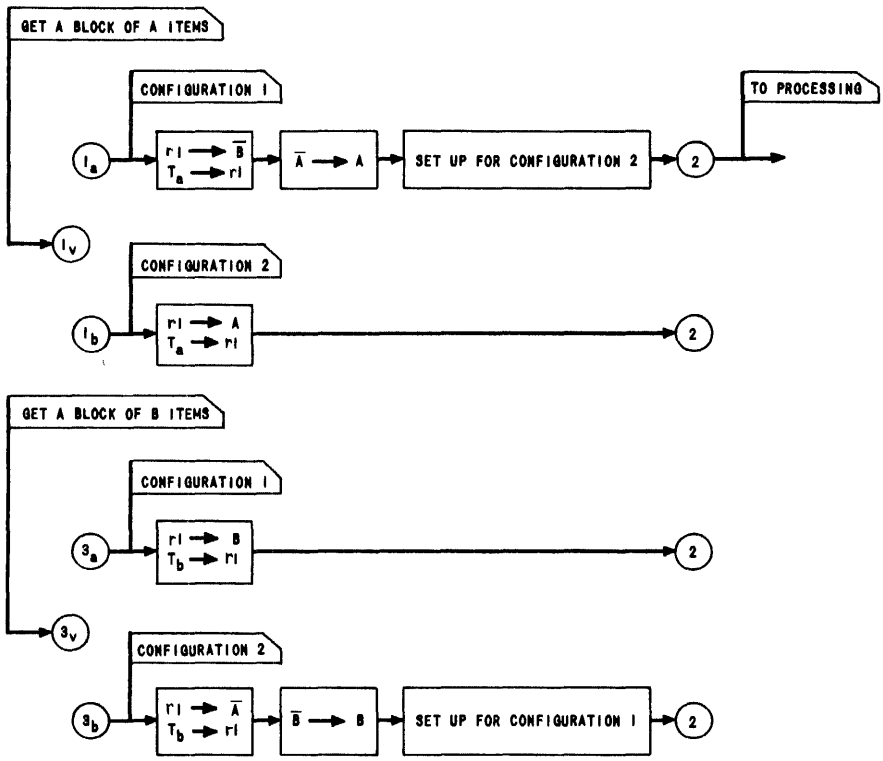


FIGURE 9-5

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 10

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 transfer neon will not be lit. If transfer of control is not indicated, the operator can cause a transfer of control by depressing a switch, called "force transfer". If transfer of control is indicated, the operator can prevent transfer of control by raising this switch.

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 switch, 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 switch, 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 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 called a 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 five-position switch on the Supervisory Control Panel. The positions are labelled one addition, one step, one operation, one instruction and continuous. Of these, only the continuous and one instruction positions are of significance here.

If the switch is on continuous, 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 and the start bar is then depressed, the computer will stop at the completion of the stage of the four stage cycle currently being executed. 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 at the end of the first stage of the cycle which it encounters. Depressing the start bar will cause the computer to complete the execution of the current stage of the four stage cycle and stop at the end of this stage of the cycle. If this process is continued; and if each time the computer stops on gamma or delta 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 instructions 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 three 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 000, a word of zeros can be transferred to CC by depressing a switch called the clear C switch Depression of the clear C switch is the operation referred to as "clear C". If the programmer wants to start the fill operation with some cell other than cell 000, 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 029, the operator could SCICR a 0 0m U0m instruction pair. The U0m instruction would specify cell 029. Then, by putting the computer on one instruction, the operator could cause the computer to execute the 0 0m U0m instruction pair. At the end of the execution the address in CC would be 029. The operator can then begin the fill operation at cell 029.

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 three 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. 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 contents of the memory on tape.
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.

chapter 11

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 characters 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 10 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.

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 any 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
		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

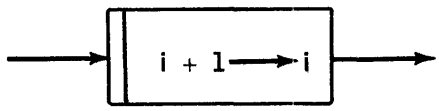
The Univac Magnetic-to-Paper Tape Converter, MTP, translates magnetic tape into the five, six, or seven level code of perforated paper tape. The MTP consists of a magnetic tape reader and a paper tape punch.

The punch operates at 60 characters per second. The MTP automatically punches teletypewriter function codes in the paper tape.

chapter 12

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^c; 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 12-2, 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.

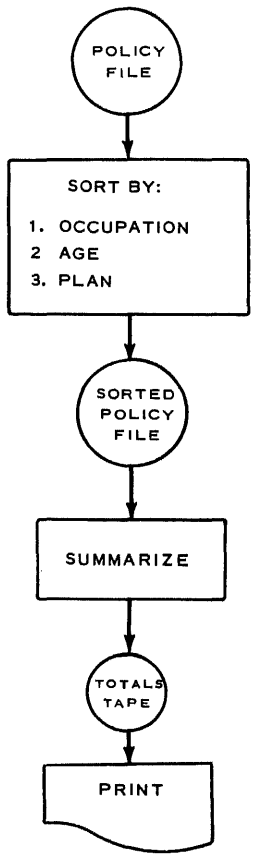


FIGURE 12-1

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 12-2

The main steps in the solution are shown in Figure 12-1. 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. The output of the sort is the sorted policy file which forms the input to the next operation which is the summarizing run. Figure 12-3 represents the essential steps in this summarization.

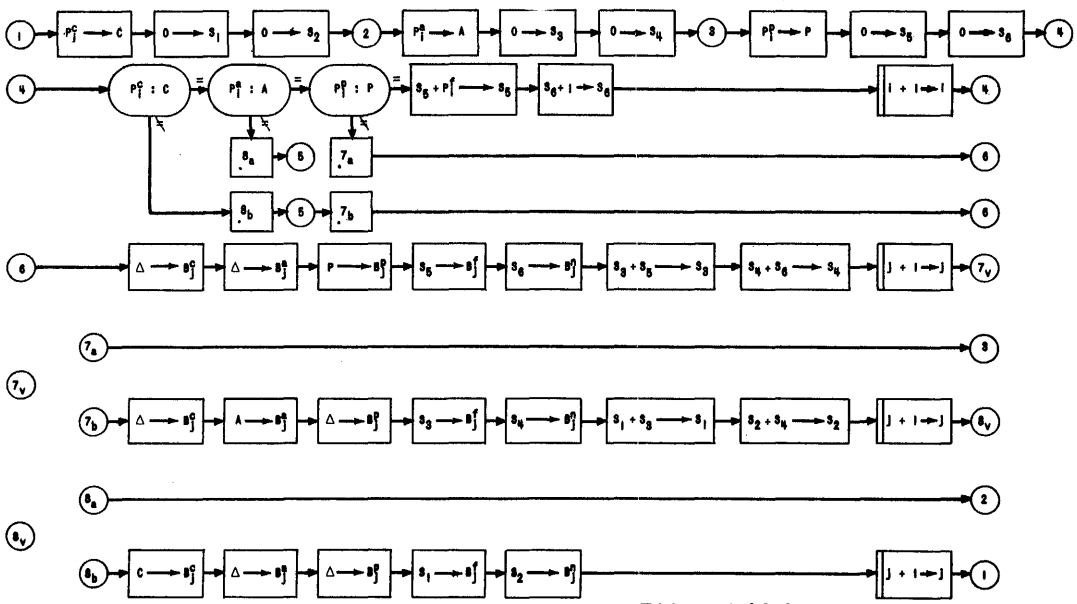


FIGURE 12-3

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 second 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 (8v))

Each output item, B_j, consists of the following five fields:

B_j^c = The occupation code field

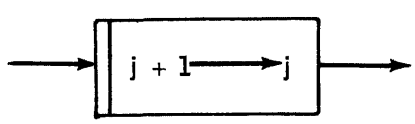
B_j^a = The age field

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, (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 (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 (7b) and (8a) are set, and control is again transferred to (6) where an output item containing the totals under P is formed as previously. In this case, since (7b) is set, an output item containing the age totals in formed. Δ's are inserted into B_j^c and B_j^p. The age, A, is inserted into B_j^a, and the totals under classification CA, S_3 and S_4, are inserted into B_j^f and B_j^n respectively. Then the age totals are added to S_1 and S_2 (the totals for class C). Connector (8a) then transfers control to (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 (7b) and (8b) are set, and control is transferred to (6) where, as shown previously, output items for the plan and age totals are formed. Then (8b) causes an output item to be formed for the totals under the classification code C. C is inserted into B_j^c; and Δ's, into B_j^a and B_j^p. S_1 and S_2 are inserted into B_j^f and B_j^n respectively; and the output operations, executed. Control is then transferred to (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 12-2.

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 12-4.

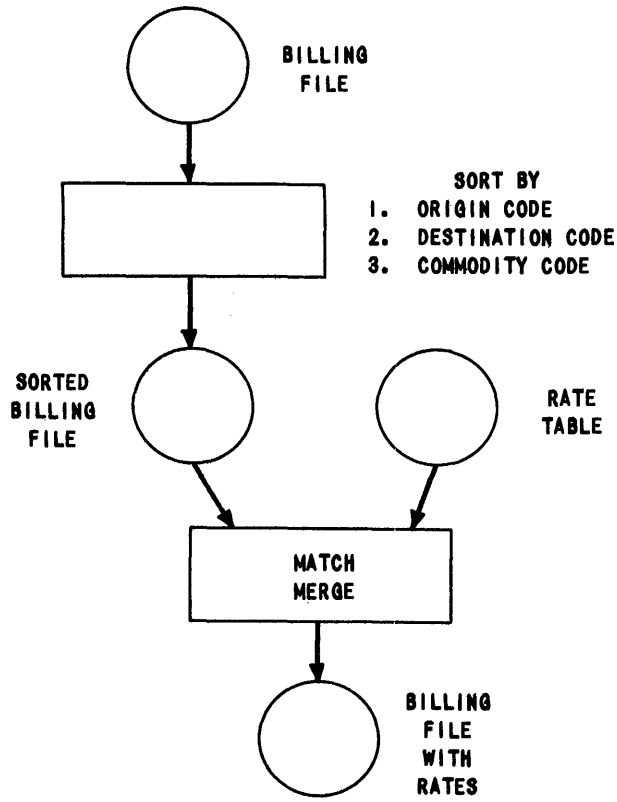


FIGURE 12-4

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 12-5.

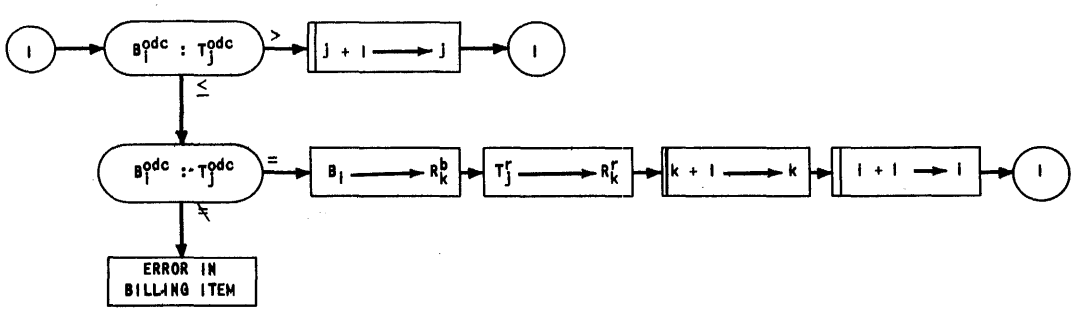


FIGURE 12-5

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 $\rightarrow$ [k+1 $\rightarrow$ k] $\rightarrow$ implies the output operations necessary to record W_k on tape, while the box $\rightarrow$ [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 12-4 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 12-6 is the required match merge necessary to select the required table entries E_n noted above.

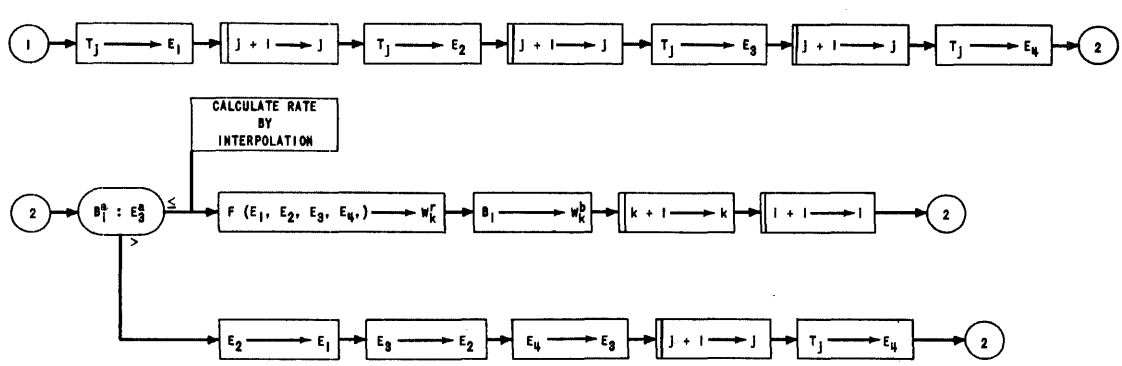


FIGURE 12-6

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_i^a. 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_i^a is compared with the argument E_3^a. If B_i^a is greater, each En is displaced down one position with E_1 being dropped and the next T_j becoming E_4. When, finally, the first E_3^a is located which is just greater than (or equal to) B_i^a, 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 → [k+1 → k] → 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_i^m
2. The part number of a part used on this model, M_i^{pn}
3. The number of such parts used on this model, M_i^n

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_j^m
2. The number of such models to be constructed, P_j^n

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 12-7 depicts the major operations required in exploding the production schedule.

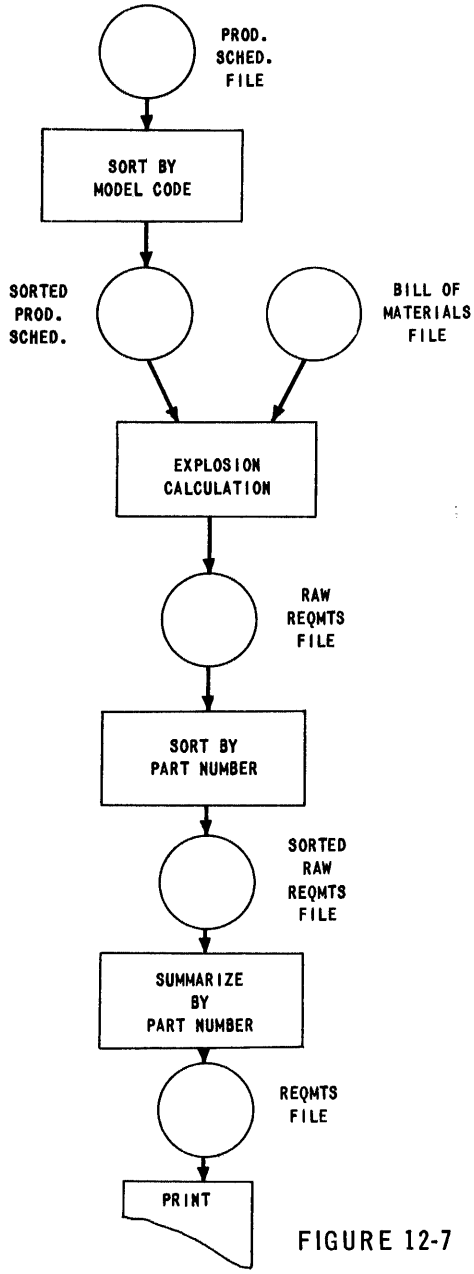


FIGURE 12-7

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 called the 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 12-8 is a flow chart showing the method of calculating the raw requirements.

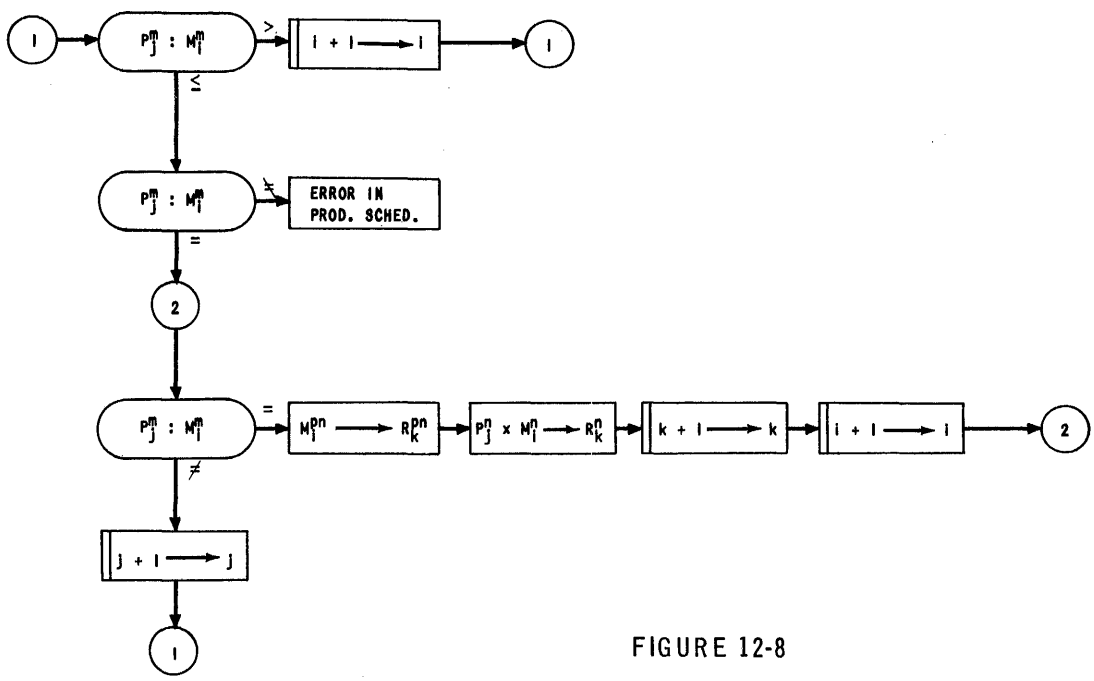


FIGURE 12-8

At (1) the model codes of the first production schedule item and the first bill of materials item are compared. If the model code called for 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 → [k+1 → k] → 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 12-9 shows the general sequence of steps required in calculating the phased requirements.

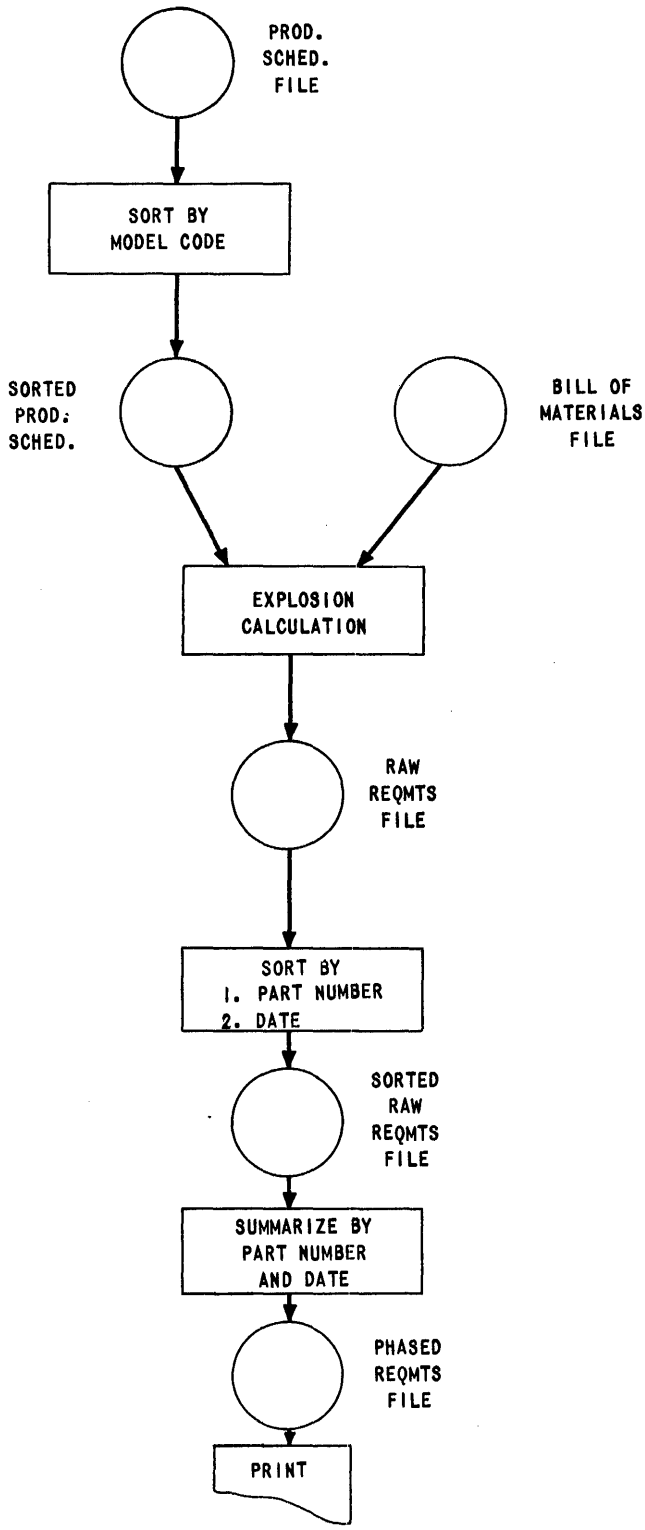

PROD.
SCHED.
FILE
SORT BY
MODEL CODE
SORTED
PROD.
SCHED.
BILL OF
MATERIALS
FILE
EXPLOSION
CALCULATION
RAW
REQMTS
FILE
SORT BY
1. PART NUMBER
2. DATE
SORTED
RAW
REQMTS
FILE
SUMMARIZE BY
PART NUMBER
AND DATE
PHASED
REQMTS
FILE
PRINT

FIGURE 12-9

The same essential steps are found in this solution as described earlier for Figure 12-7. Of course, the explosion calculation will necessarily be different. The flow chart of this explosion run is shown in Figure 12-10.

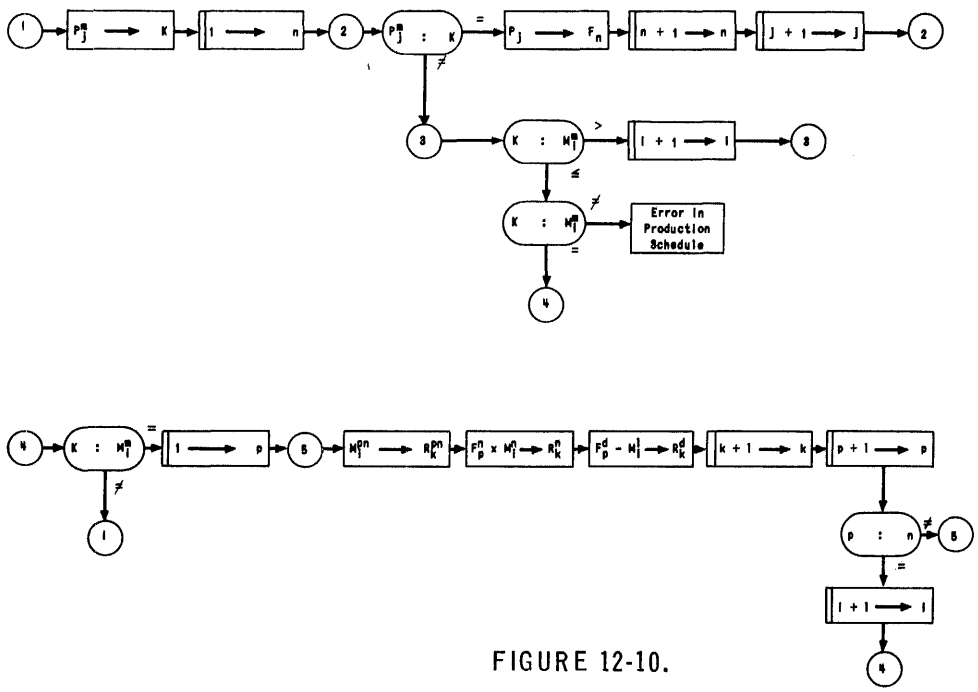


FIGURE 12-10.

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 $\boxed{k+1 \rightarrow k}$ implies the output operations necessary to record the item R_k on the raw requirements tape. The box $\boxed{p+1 \rightarrow p}$ 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 13

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 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 but wrong" 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.

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 13-1, 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 13-1

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,

```
010  B0 880
                A- 881   }  addition
011  H0 882
                S- 880
012  L0 881                }  check
                Q00020
```

If control is transferred to cell 020, the addition was correct; if control passes to cell 013, 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 (O-E 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 checker 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.

In addition to the O-E checks on transfers, the regular operation of the computer is interrupted every five seconds for the Periodic Memory Check, PMC. During PMC the contents of the memory are read into the HSB O-E Checker. Should an even count be registered for any of the 12,000 characters the HSB O-E Checker will alert the error circuitry and stall the computer. PMC prevents a faulty character from going undetected for long periods of time and possibly dropping enough pulses to pass the odd-even check.

However, there are failures that the odd even check cannot detect. For this reason duplicate 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 cycling unit, which keeps track of the stage of the four stage cycle that the computer is on,

and 10. the Time Out circuits, which determine whether Univac is on TO or Time On.

LOGICAL CHECKS

In addition to the duplicated circuits and odd-even checkers there are a large number of internal logical checks designed to further insure error free computation. Logical checks are employed wherever it is not feasible to duplicate equipment or where no data transfer is involved to make use of odd-even checks.

TANK SELECTOR CHECKER

This checker is a check on the fourth and fifth instruction digit set-up of SR. A further word of explanation is necessary for this checker. There are two general types of checking circuits:

1) In a negative checker the error neon is lit when an error is detected.
2) In the positive type checker, the error neon is lit first and only correct operation will extinguish the neon in time to prevent stalling the computer.

This is a positive type checker. If the upper tank selector neon is lit and the computer is stalled it means the fourth instruction digit was set up incorrectly, if the lower neon is lit and the computer stalled it indicates that the fifth instruction digit is incorrect. It is quite possible to have a Tank Selector Error through a faulty program. An instruction B00A12 for example, will show a fourth instruction digit error. The sixth instruction digit set up is checked by circuits which compare the check pulse in SR against a computed check pulse.

FUNCTION TABLE INTERMEDIATE CHECKER

The Function Table Intermediate Checker is a check that the first instruction digit was set up correctly in the Static Register. This checker also acts as a shift selector check.

FUNCTION TABLE OUTPUT CHECKER

This is a duplicated positive type checker, whose function it is to check on the proper execution of instructions.

TAPE CHECK

Along with the seven information bits recorded on tape for each character, an eighth bit called the Sprocket Channel Pulse is also recorded. When information is being read from tape the Sprocket Channel Pulse indicates the presence of a character and actually initiates the process of synchronizing the incoming information with the timing of the computer. If a Sprocket Channel Pulse is not read from tape along with information pulses the Tape Check Error neon is lit.

INPUT-OUTPUT INTERLOCK CHECKER

The input-output interlock circuits are set at the beginning of any input-output operation and are reset when that operation has been completed successfully. When the interlocks are set, further orders of the same type or using the same equipment are prevented from being executed. It is essential to correct operation of the computer that the interlock circuits function properly. This checker, when set, indicates one of the following failures:

1) The read interlock failed to set at the beginning of the last read order.
2) The write interlock failed to set at the beginning of the last write order.
3) The Uniservo in question was set to execute a backward read when a forward read was ordered.

INPUT SYNCHRONIZER > 720 CHECKER

Digits are recorded serially along the tape, and are thus picked up one at a time when the tape is read. The computer counts the number of digits read and after the 720th digit (last digit of the 60th word) has been read and the space between blocks is encountered the read is terminated. Through a failure in the input-output control or photocell circuits a short (less than 720 digits) or a long (greater than 720 digits) block may be encountered. Either of these two cases lights the > 720 error neon.

As far as the computer is concerned, a short block is defined as a read of 59 complete words and at least one, but less than twelve more digits, followed by the space between blocks. The Uniservo will then stop reading tape and set the > 720 error. A long block occurs when the computer reads a full 60 words and at least one more digit before encountering a space between blocks. The tape stops in the next space between blocks or photocell area, whichever is first, and sets the > 720 error. In either case setting > 720 error prevents the next read order or Supervisory Control input or any other order affecting the Uniservo causing the error from being executed.

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.

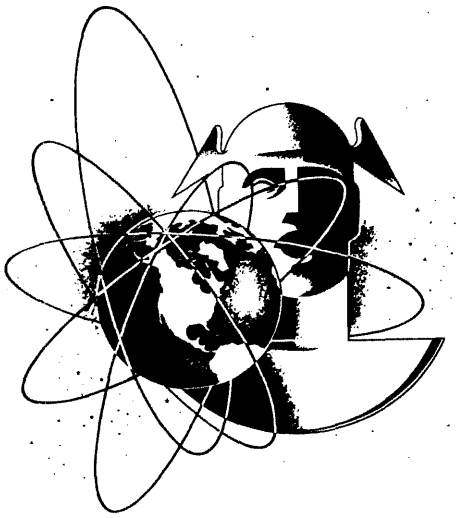
BUILT IN CHECKS ON THE PROGRAM

Besides checking the accuracy of its operations, the computer also checks for the occurrence of an adder-alph error, in which case, the computer immediately stalls and lights an appropriate neon on the Supervisory Control Panel.

UNIVAC I TAPE CHART

TAPE USE	SPACING IN INCHES	PULSE DENSITY/ CHAR/IN.	BLOCKS PER REEL (*BLOCKETTES)	UNIVAC I READ-WRITE TIME PER BLOCK	PER REEL	REWIND TIME PER REEL	FEET UTILIZED PER REEL
Univac I	5.625 / block 2.4 between blocks	128	2,275	105.25 ms	3.99 min.	3.04 min.	1521
Unityper II Verifier	2.4 / blockette 2.4 between blockettes 2.4 between blocks	50	500 *	313.0 ms	26.08 sec.	24 sec.	200
Card-To-Tape Converter	.9375 / blockette 1.8 between blockettes 2.4 between blocks	128	6,400 *	195.25 ms	3.47 min	3.03 min.	1513
Uniprinter	36.0 / block 2.4 between blocks	20	475	409.0 ms	3.24 min	3.04 min	1520
Tape-To-Card Converter	.9375 / blockette .1 between blockettes 2.4 between blocks	128	12,900 *	110.25 ms	3.95 min	3.06 min	1527
High-Speed Printer	.9375 / blockette 1.0 between blockettes 2.4 between blocks	128	8,400 *	155.25 ms	3.62 min	3.04 min	1520

UNIVAC I Start-Stop Time : 44 ms; Interlock: 5 ms; Tape Speed: 100" / Second.



UNIVAC®—The FIRST Name in Electronic Computing Systems

U1592 Rev. 1A PRINTED IN U.S.A.