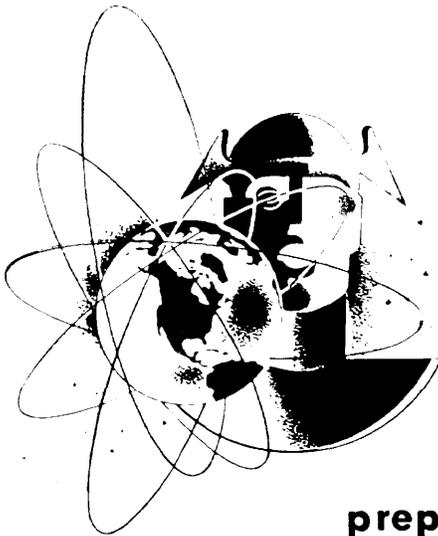


class notes

C O U R S E

0 0 1

INTRODUCTION TO COMPUTERS



prepared by the Training Section,
Electronic Computer Department

Remington Rand

SECTION 1

Components of a Computer

The Univac is an electronic computer designed to perform repetitive clerical and mathematical computations at a high rate of speed and accuracy. In order to make clear the function of each component of the computer and their interrelations let us consider the operation of a payroll clerk.

The clerk has a stack of time cards listing the hours worked by each employee of the company. She has a second stack of cards that contains the hourly rate of pay, number of dependents, and other fundamental information about each of the employees.

To aid her in her work she has a desk calculator which enables her to do addition, subtraction, multiplication and division. A ledger book is also provided for entering the desired pay data as it is computed.

In addition to the above elements, the clerk provides a supervisory element: She selects a time card and hourly rate card for each employee and by means of the calculator multiplies hours by rate to obtain a gross pay, subtracts from this his withholding tax, etc., and enters the net pay in her ledger.

This payroll process can be thought of as requiring four elements: input, output, arithmetic, and supervision. As shown in Figure 1, the time cards and hourly rate cards are the input, the calculator accomplishes the arithmetic, the ledger entries are the output, and the clerk is the supervision.

When we examine further the role of the clerk in this operation, we find that her ability to turn the input of time and rate cards into the ledger entries of net pay depends upon her remembering the individual steps of the operation, her instructions and the order of their execution. That is, she must remember that she is to subtract the withholding tax from the gross pay, and that this tax is computed by multiplying the number of dependents by \$13.00 and subtracting this from the gross pay, multiplying the difference by 20%, and so forth. Her supervision then actually consists of two functions, memory and control. This memory function becomes even more evident if the clerk has other duties to perform as well. She may be required to maintain the rate cards: for example, changing a man's rate of pay when he is given a raise or adding or removing dependents upon notice.

When we try to mechanize this process, we have the option of building the supervision into the device so that it will do only payroll calculations, or of building into it the ability to do a small number of fundamental operations and then storing in a memory the proper sequence of performing these operations. A computer with a built-in supervision is called a Special Purpose Computer, while a computer that stores its supervision in a memory and which allows that supervision to be easily altered is called a General Purpose Computer.



FIGURE 1

The Univac is such a General Purpose Computer. Its five components and their interrelations are shown in Figure 2. The memory occupies a central position. It holds for processing the data coming from the input unit. It also holds the instructions that tell what to do with this data. The control unit selects each instruction from the memory in proper sequence and executes it. Instructions might call for the hourly rate and total hours worked to be extracted from the memory and delivered to the arithmetic unit for multiplication, the product being returned to the memory. As the net wages are computed, they are sent to the output unit for permanent recording.

In order to secure high operating speeds, these units are of electronic construction; and to assure complete accuracy, many of the units are duplicated and cross-checked. Other checking means are used when it is not feasible to duplicate equipment.

There is yet another way in which computers may be categorized which is intimately related to the concept of memory. The basic element being processed by a computer is a number. We are all familiar with two ways of representing numbers:

1. The digital method, possibly the oldest, uses a unique symbol or mark to represent each number. For example, we use the Arabic symbols with the position concept when we say that a dozen is "12" and a dozen dozen or gross is "144".
2. The analog method is familiar to us in the slide rule where numbers are represented by different distances along a stick, or by an ammeter which represents the number of amperes of current flow by the angle of a pointer. Other commonly used representations are voltage levels, rotation of a gear or shaft, and extension of a spring.

Computers may then be of two general types: Special Purpose or General Purpose, depending on whether they have a stored program of alterable instructions or not. Within each type a computer may be Digital or Analog depending upon the representation of numbers.

The Univac is a General Purpose Digital Computer.

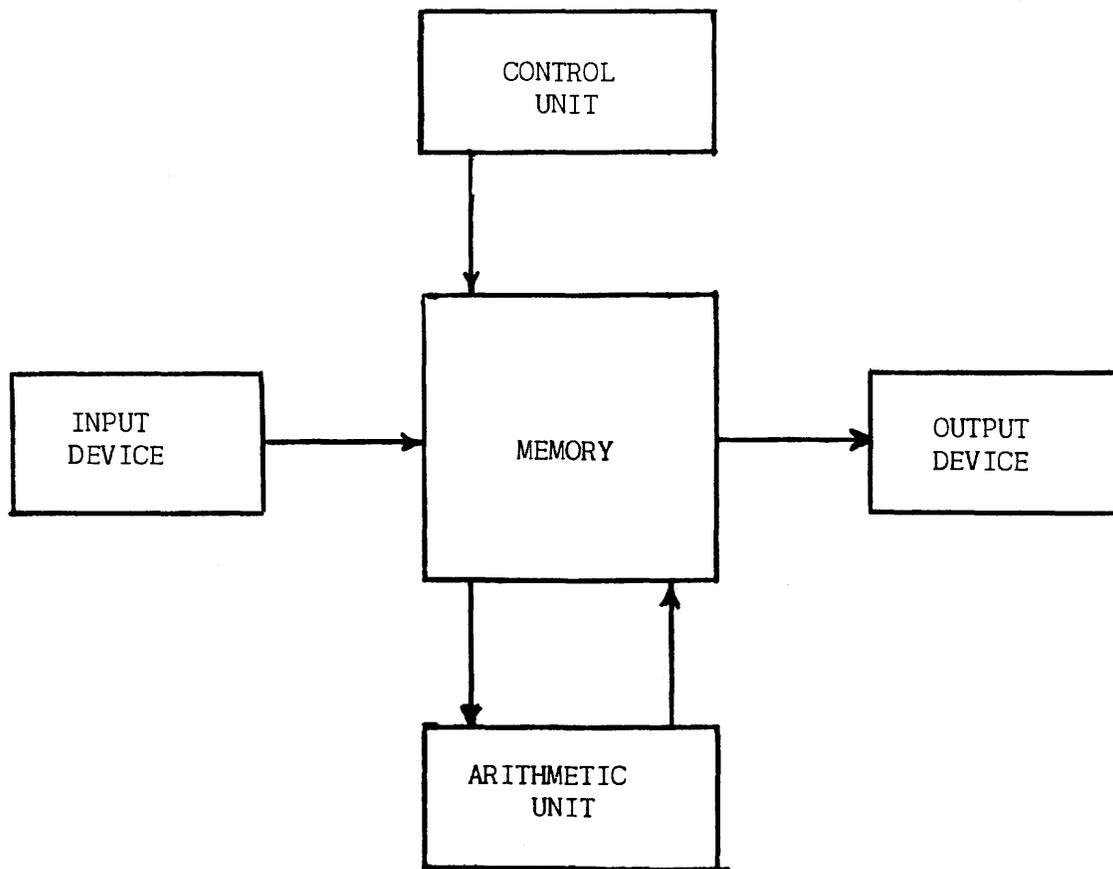


Figure 2

SECTION 11

The Memory Unit

In order to understand the operation of the Univac, we will start with the central unit of the five units: The memory. The physical make-up of the memory will be discussed in Chapter 8, however, we will now describe the general characteristics which concern the user of a computer.

The basic unit of memory in the Univac is the "word". A word always consists of twelve characters or digits. There 63 possible characters available; any combination of twelve of these constitutes a word. The 63 characters available are shown in Figure 3, but for our immediate needs we need only the numbers 0, 1, 2,...9 and the letters of the alphabet A,B,C,...Z and the dash symbol (-). Here are some typical words:

012345678906
-00121169876
ABCDEFGHI986HJ
JOHNNY-JONES

For ease of reference the digit positions of a word are numbered from left to right. Thus, in the last example above, the character in digital position 7 is the dash.

The memory of the Univac has the capacity to "store" or "remember" 1000 such words. The memory may be visualized by picturing 1000 boxes, and in each box is a slip of paper with a Univac word written on it (figure 4). In order to locate a particular word we number or "address" the boxes. In Univac the boxes are numbered consecutively from 000 to 999. Thus, in Figure 4, we can speak of the word in box 001 as being JOHNNY-JONES.

When a word represents a number, the character in digit position 1 is considered as the sign of the number. The minus sign is the dash symbol while the plus sign is a zero.

-61321542013
061321542013

Further, the computer considers all numbers to be less than one in magnitude. That is, it assumes a decimal point between digit positions 1 and 2.

A later chapter will describe how numbers larger than one can be represented.

As mentioned in Section 1, a General Purpose Computer stores its instructions in a memory. In the Univac these instructions are stored in the same 1000 word memory as the data they are to operate upon. A Univac instruction consists of six digits designated from left to right as "first instruction digit", "second instruction digit"... "sixth instruction digit". The fourth, fifth, and sixth instruction digits generally are the address of some memory box, and the first and second instruction digits indicate what operation is to be done on the word at that address. The instructions will be described in Chapter 2.

Since an instruction is six digits long, a Univac word can contain two instructions. Thus, when a word represents instructions, The left six digits (digital positions 1-6) are called the Left Hand Instruction, while the right six digits (digital positions 7-12) are called the Right Hand Instruction.

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|----|---|---|
| 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 | % | = |  |

63 Characters Representable
In The UNIVAC

Figure 3

SIMPLIFIED VISUAL CONCEPT
OF UNIVAC MEMORY

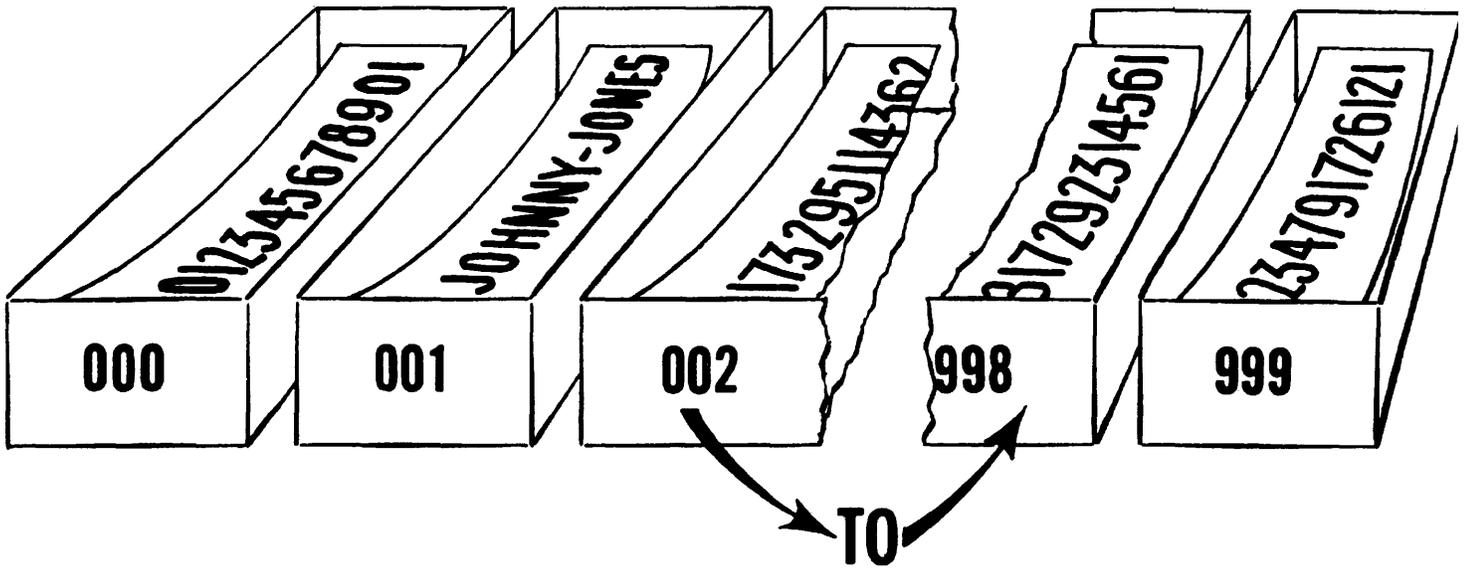


Figure 4

SECTION 111

The Arithmetic Unit

The arithmetic unit consists of an algebraic adder, a multiplier-divider, a comparator, and several special memory boxes called registers:

Register A, rA, which holds one word
" L, rL, " " " "
" F, rF, " " " "

Their interconnections are shown in Figure 5. As indicated in the drawing, the results of any arithmetic calculation (the output of the adder and multiplier-divider) are always placed in rA. In fact, it is in these and other registers to be described that the data processing actually takes place. The memory serves simply as a storage for data and instructions until they are needed for processing.

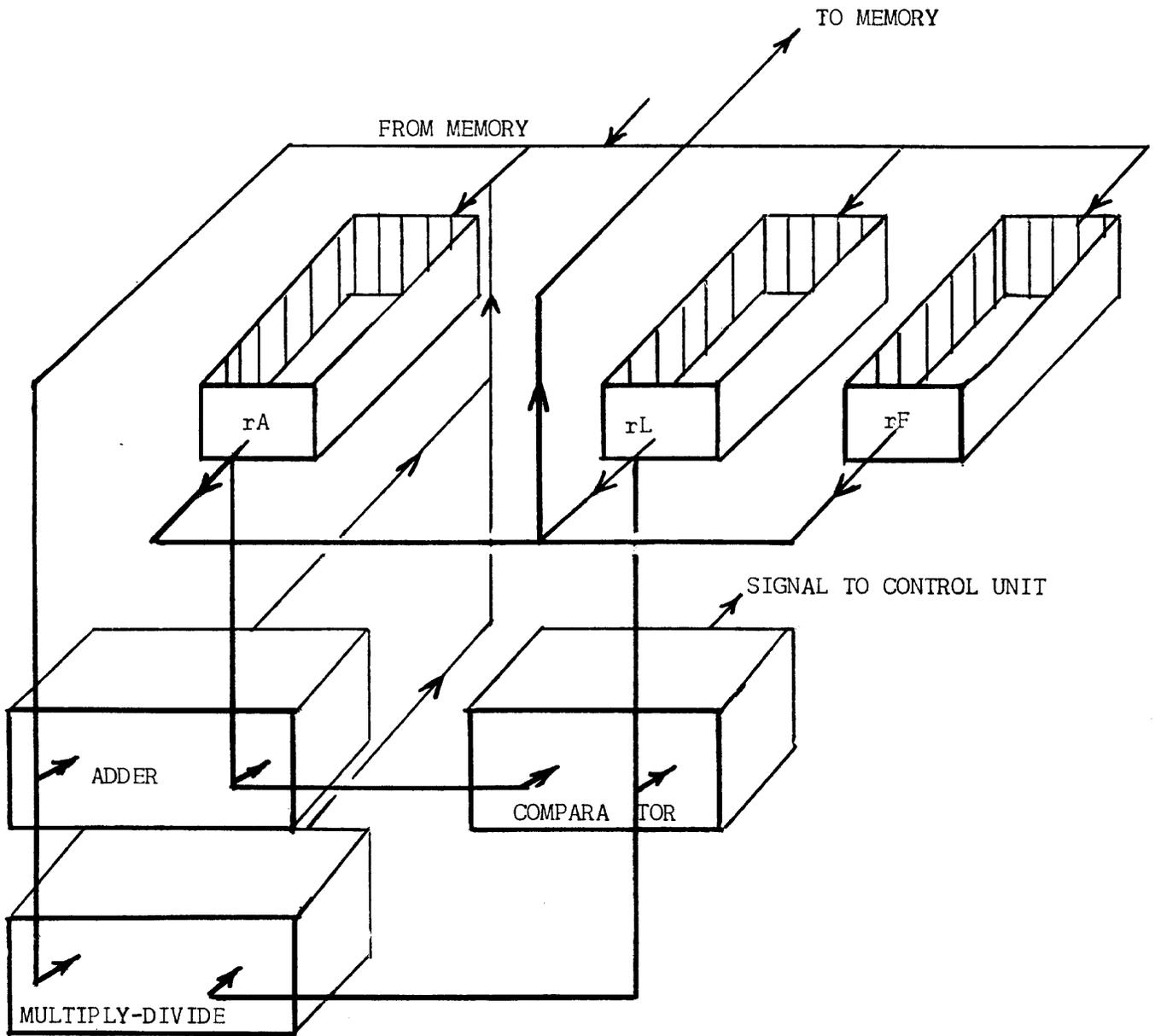


Figure 5

SECTION 1V

The Control Unit

The control unit selects the instructions placed in the memory and executes them in their proper order. The control unit consists of synchronizing and effector devices and three registers:

- The Control Counter, CC, which holds one word
- The Control Register, CR, which holds one word
- The Static Register, SR, which holds one-half word

Figure 6 depicts the interconnections between these registers. The word in the Control Counter always has the following appearance:

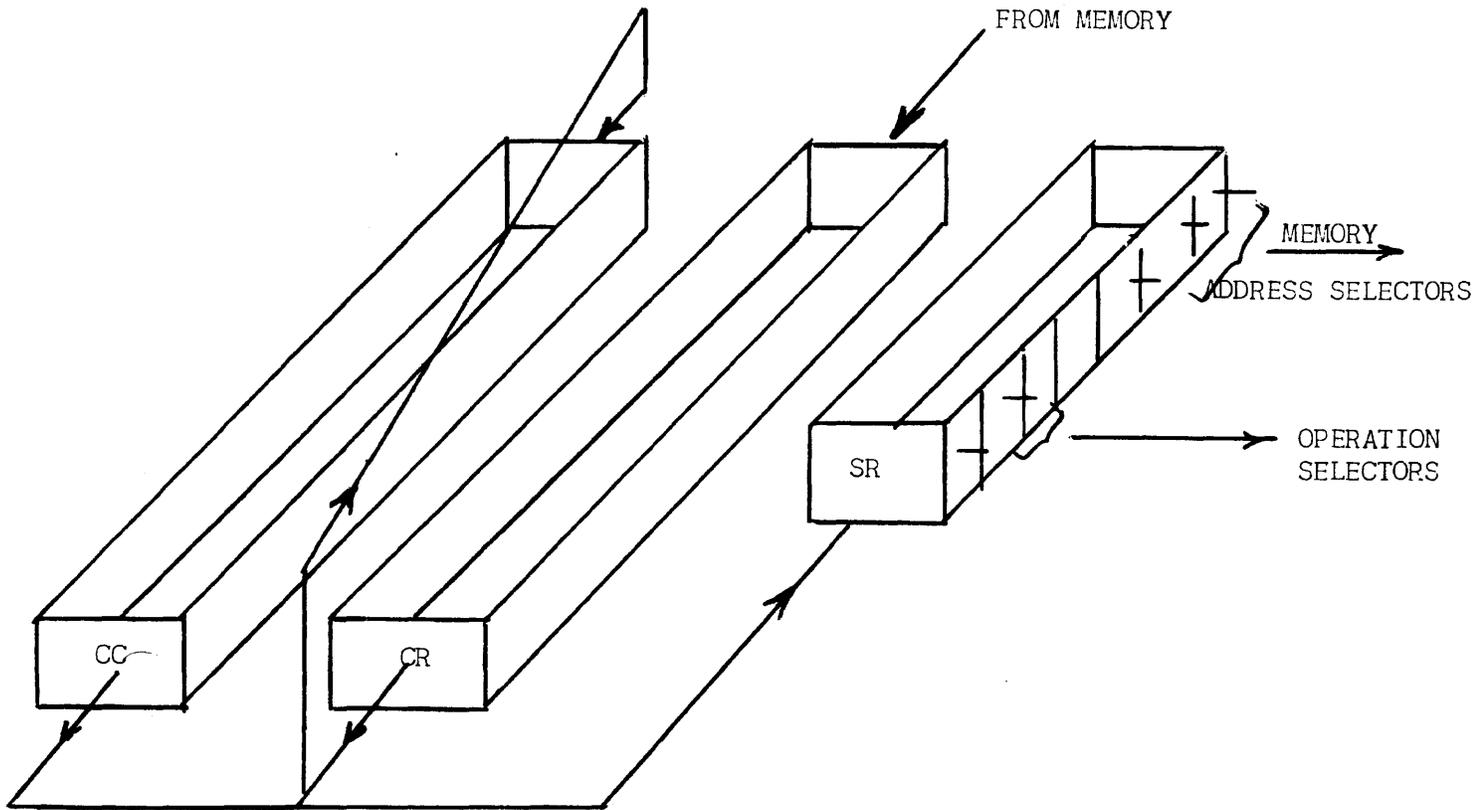
00000000XXX

where XXX is some number between 000 and 999. This number is the address of the next pair of instructions to be executed. A computer which stores its instructions in a memory must first locate and extract from this memory the instruction and then execute it. Since the UNIVAC stores two instructions per word, each memory look-up selects a pair of instructions. The Control Register is used to hold one instruction while the other is being executed. 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:

| <u>Step</u> | <u>Description</u> |
|-------------|--|
| 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 . As we shall see, there are ways of breaking this pattern of executing instructions from sequential addresses.

Figure 6



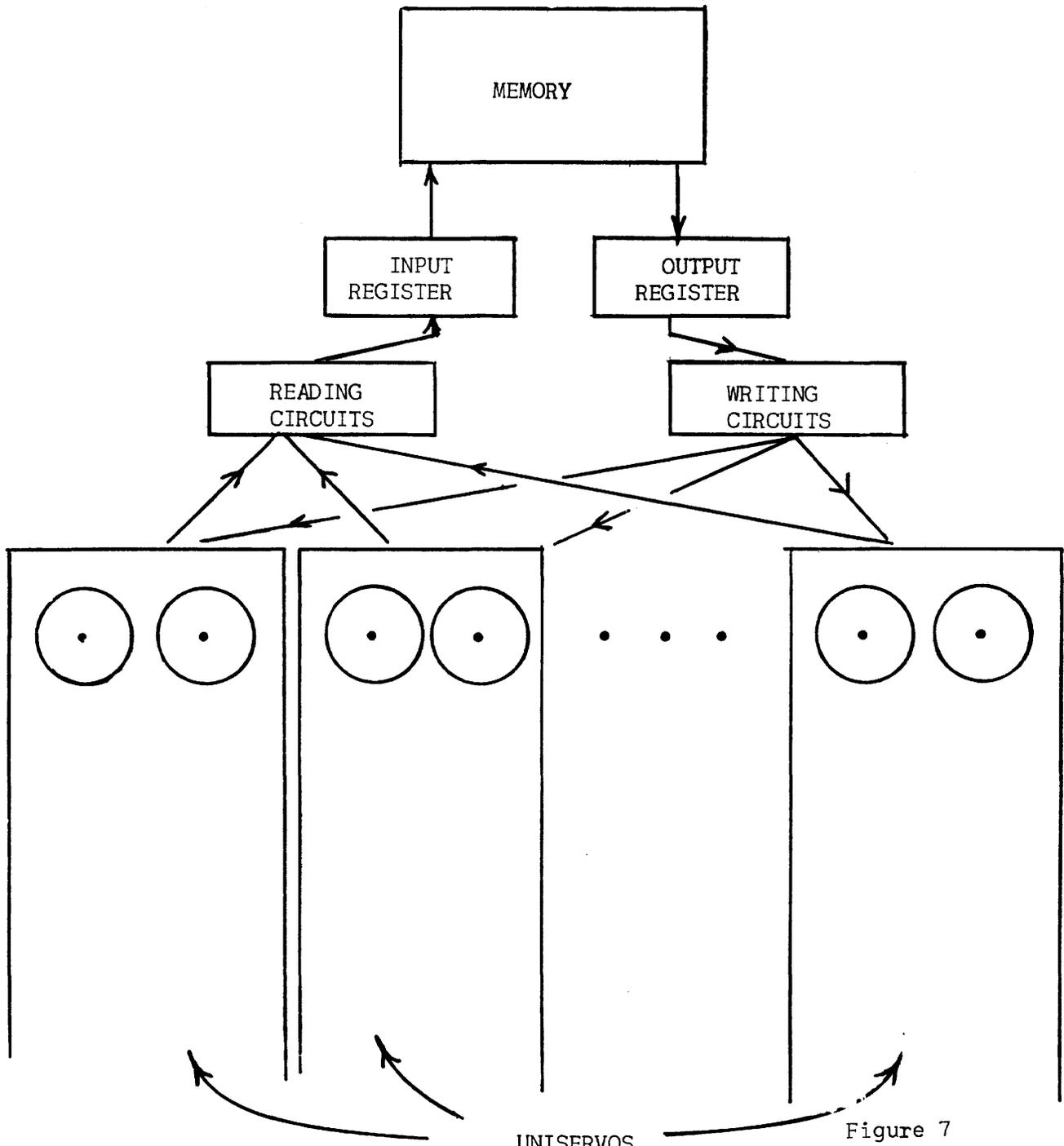
SECTION V

Input and Output Units

Commercial calculations almost always involve vast amounts of data, volumes far in excess of the storage capacity of the main memory of any computer. In the Univac System, volume storage of data is achieved through magnetic recording on metallic tape. Initial transcription of data onto magnetic tape is achieved through terminal equipment (Unitypers, Card-To-Tape Converters) to be described later. The input of the Univac consists of a number of magnetic tape units called Uniservos which will read the information recorded on tape into an input register, rI, where it may then be transferred into the memory. These same Uniservos will also record information on metallic tape and serve as output units. Uniprinters, Tape-To-Card Converters, and High-Speed Printers serve as terminal equipment on the output end. These devices, also to be described later, read magnetic tape and produce a visible or hard copy of the information. Data in the memory is first transferred to an output register, rO, from which the Uniservos operate. Thus, only small amounts of data need be brought into the main memory at one time for processing. After calculation, the results are recorded on tape and the original data replaced by new data tape.

Figure 7 shows the interconnections between memory, input, and output units.

While magnetic tape forms the main input-output media, occasions arise where directly intelligible output and input is desirable. An electric typewriter, called the Supervisory Control Printer, connected directly to the computer and operating under program control, permits limited volume direct output. Similarly, a keyboard on the control console permits small volume direct input to the computer under program control.



UNISERVOS

Figure 7

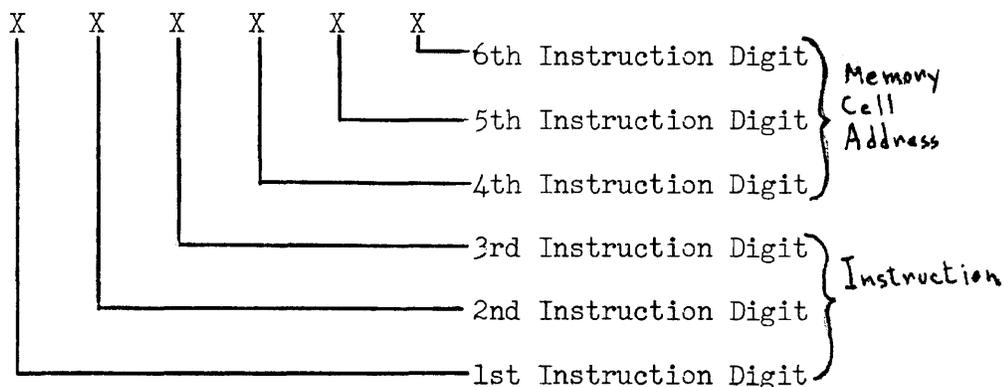
SECTION 1

Introduction and Notation

The Univac System responds to 43 basic instructions which are summarized in the Appendix of the Univac Programming Manual. These instructions were selected after a detailed study of many data processing problems to determine the most desirable and efficient set. The complete code with the unique characteristics of many of the instructions provides the skilled coder a highly efficient and flexible means of problem solution.

Our purpose in this course, however, is solely the development of your comprehension of computers - as to what problems can be solved on them and a general understanding of how the solution is prepared. For this reason, we shall consider a restricted set of instructions. For many of these instructions we shall describe only their main characteristics for the sake of simplicity. As each group of instructions are described, sample problems will be coded to illustrate their use. Further problems are included as student exercises; solutions of these problems will be found in the Appendix.

Chapter 1 described the Univac instruction as consisting of 6 digits. The digits were labeled as shown in the figure below:



In the general case, the 4th, 5th, and 6th instruction digits are a memory cell address, while the 1st and 2nd instruction digits specify what operation is to be done to the word at that address. The 3rd instruction digit is available for a possible increase in memory size. We shall use the letter "m" to designate the 4th to 6th instruction digits in the description of the instructions. The reader should bear in mind that the configuration of characters called an instruction in the instruction lists to follow BECOMES an instruction when they are in the Static Register during the γ or δ stages. At any other time they have no significance as instructions.

SECTION 11

Instruction List A

B00 m, This instruction, when executed by the computer, causes the word in memory cell m to be duplicated in rA. The former contents of rA are erased. The word will still remain in m, unaltered. This is true of all instructions reading information from the memory or the special registers.

H00 m, This instruction, when executed by the computer, causes the word in rA to be duplicated in memory cell m. rA remains unaltered, but the former contents of m are, of course, erased.

A00 m, This instruction, when executed by the computer, causes the word in m and the word in rA to be sent to the adder, the sum being stored in rA. The initial contents of rA will be destroyed, but the contents of m will remain unaltered.

S00 m, This instruction is similar to the add instruction above, except the contents of m are subtracted from rA.

500 m, ⁵⁰ This instruction causes the computer to print on the Supervisory Control Printer the word contained in cell m. The contents of m are not altered.

5 INSTRUCTION WILL WRITE ON TAPE 120 PULSES/INCH.

900 m, This instruction causes the computer to stop executing instructions. Memory address m is ignored. 90 = STOP

It is desirable at this point to code a simple problem to illustrate these instructions:

Memory location 100 contains the on hand amount of a certain stock item. Memory location 101 contains the on order amount, while cell 102 gives the expected requirements for the next sixty days. Compute (on hand) + (on order) - (required), print and stop.

It is convenient to start the instructions in cell 000, though any other location (except 100-102) will do. In this case the solution is:

| Memory Cell | Contents | | Remarks |
|-------------|------------|-------------|---|
| | Left Inst. | Right Inst. | |
| 000 | B00 100 | A00 101 | on hand → rA rA + on order → rA |
| 001 | S00 102 | H00 103 | rA - required → rA Store rA for printing |
| 002 | 500 103 | 900 000 | Print Stop |

For convenience in writing remarks, the left and right hand instructions are written on alternate lines, though if we could "look" in memory cell 000, for example, we would see B00100A00101. As a further convenience in writing down instructions, we never write the third instruction digit for any instruction, nor the second instruction digit when it isn't necessary.

In the B, H, A, and S instructions we needn't specify the second instruction digit and would write them as B 100 for example. This is solely to save effort in writing the instructions, these omitted digits will be inserted by the operator when the instructions are placed in the memory.

Student Exercises

1. Memory locations 100-104 contain a list of receipts; print their total and stop.
2. The following quantities are stored in the designated locations:

100: A
101: B
102: C
103: D

Compute and print $2A - B + 3(C + D)$ and then stop the computer.

SECTION 111

Instruction List A, Continued

L m. This instruction duplicates in rL, the contents of memory cell m.

M m. This is the multiply instruction. The words in m and rL are sent to the multiplier and the rounded product is stored in rA. The former contents of rA are destroyed.

D m, This instruction sends the contents of m and rL to the divider, the rounded quotient, $(m) \div (rL)$, is stored in rA. The former contents of rA are destroyed.

You will recall, in Chapter 1, we noted that Univac considers the decimal point to lie between digit positions 1 and 2, and that digit position 1 is the sign digit. This means that as far as the computer is concerned, every number lies between 1 and -1. How then, can we represent numbers larger than one?

No problem is involved when addition or subtraction are the only arithmetical operations involved, as we may assume a decimal point at any desired position in the word. For example, if we let the inverted carat \wedge indicate the assumed decimal point, the assumed decimal point of a sum or difference will be in the same position as for its factors:

| | | |
|---------------|------------------------|------------------------|
| \$3600.05 | 036 000 500 000 | 000 000 360 005 |
| <u>156.23</u> | <u>001 562 300 000</u> | <u>000 000 015 623</u> |
| \$3756.28 | 037 562 800 000 | 000 000 375 628 |

But for multiplication:

\$15.32 x 2.5 = \$38.30

015 320 000 000 x 002 500 000 000 = 000 383 000 000

The assumed point is not in the same position for the product as for the factors. A similar result is also true for division. A simple set of rules tells us where the assumed decimal point of a product or quotient lies when the assumed decimal point of its factors are known.

Rules for Positioning Decimal Points

Rule for Multiplication: If the assumed decimal point lies m places to the right of the machine decimal point in one factor and n places to the right in the other, the assumed decimal point of their product will be m + n places to the right of its machine decimal point.

If the assumed decimal point of one or both factors is pegged m (or n) places to the left of the machine decimal point, treat m (or n) as negative and apply above rule. If the sum m + n is negative, its interpretation is that the assumed decimal point lies that many places to the left of the machine point.

Rule For Division: If the assumed decimal point of the dividend lies m places to the right of the machine point while that of the divisor lies n places right, the assumed point of the quotient lies m-n places to the right of the machine point. Here again negative m, n, or m-n are interpreted as meaning the assumed point lies that number of places left of the machine point.

EXAMPLES

$$A = \overset{+}{\underset{\wedge}{XX}} XX X XXX XXX$$

$$B = \overset{+}{\underset{\wedge}{X}} X XXX XXX XXX$$

$$A \times B = \overset{+}{\underset{\wedge}{XX}} XXX \cdot XXX XXX$$

$$A \div B = \overset{+}{\underset{\wedge}{XX}} X \overset{+}{\underset{\wedge}{XX}} XXX XXX$$

$$A = \overset{+}{\underset{\wedge}{XX}} XXX XXX X \overset{+}{\underset{\wedge}{XX}}$$

$$B = \overset{+}{\underset{\wedge}{XX}} XXX XXX XXX$$

$$A \times B = \overset{+}{\underset{\wedge}{XX}} XXX XXX \overset{+}{\underset{\wedge}{XXX}} XXX$$

$$A \div B = \overset{+}{\underset{\wedge}{XX}} XXXXXX \overset{+}{\underset{\wedge}{XX}} X$$

Student Exercises:

Indicate Position of Assumed Decimal Point

1. $A = \overset{+}{\underset{\wedge}{XX}} XX XXX XXX XXX$

$$B = \overset{+}{\underset{\wedge}{XX}} XX XXX XXX XXX$$

$$A \times B = ?$$

$$A \div B = ?$$

2. $A = \overset{+}{\underset{\wedge}{XX}} XXX XXX X \overset{+}{\underset{\wedge}{XX}}$

$$B = \overset{+}{\underset{\wedge}{XX}} XX XXX XXX XXX$$

$$A \times B = ?$$

$$A \div B = ?$$

3. $A = \overset{+}{\underset{\wedge}{XX}} XXX \overset{+}{\underset{\wedge}{XX}} X XXX$

$$B = \overset{+}{\underset{\wedge}{XX}} X \overset{+}{\underset{\wedge}{XX}} XXX XXX$$

$$A \times B = ?$$

$$A \div B = ?$$

SECTION 1V

Instruction List B

00 m, This is the skip instruction. It tells the computer to pass on to the next instruction altering neither the memory or registers.

U m, This instruction is often called an unconditional transfer of control. As mentioned in Chapter 1, this is one of the instructions that break the sequential execution of instructions. The U m instruction tells the computer to begin executing instructions sequentially beginning with the instruction pair in cell m. The memory and arithmetic registers are not affected.

In detail, the U m instruction causes the memory address digits of the Control Counter to be replaced by the address m. The U m instruction must be a Right Hand Instruction to be executed as described. This is true only of the U, Q, and T instructions (see below). All other instructions work equally well as Left or Right Hand Instructions.

Q m, This is one of two conditional transfer of control instructions. The U m instruction always transfers control, but the Q m instruction transfers control only upon certain conditions. If the contents of rA and rL are identical, the computer interprets the Q m instruction as though it were a U m. If rA is not equal to rL, the Q m is interpreted as a skip. This instruction then allows a choice between following one set of instructions rather than another when that choice can be expressed by the equality of two numbers (or alpha-numbers).

T m, This is the other conditional transfer of control instruction. This instruction acts similarly to the Q m instruction, except that the condition for transfer is now that rA be algebraically larger than rL. A few examples will illustrate the principle.

| rA | rL | Does T m 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 |

Since letters as well as numbers may be present in a word, we need to know their relative weights (often called the collating sequence). In the comparison, the least significant column of a word is digit position 12. The significance increases as we move to the left until we reach the most significant column, digit position 1. Bearing this in mind, for a given column, the order of magnitude for the 63 Univac characters shown in Figure 3 of Chapter 1 is as follows: for any character in the chart, all characters to its right in the same row and all the characters in the rows below it are larger, while all characters to its left in the same row and all characters in the rows above it are smaller. Thus, the i (ignore) symbol is the smallest Univac character. Again, the following examples will illustrate the principle:

| rA | rL | Does T m Transfer Control |
|-----------------|-----------------|---------------------------|
| ABC DEF GHJ KIM | 123 456 789 ABC | Yes |
| 111 111 111 AZ9 | 111 111 111 BZZ | No |
| -AB 904 6DE FG7 | 000 123 456 789 | No |
| -AB 904 6DE FG7 | -12 345 678 9AB | No |

Example Problem #1

FICA CALCULATION

A year-to-date total of FICA taxable earnings, FE, is stored in memory cell 100. A year-to-date FICA tax paid, FT, is stored in cell 101. This week's pay, P, is stored in cell 102. Compute the FICA tax, T, for this week and print, bringing the totals up-to-date. Assume that the assumed decimal points are FE = OXX XXX XXX XXX, FT = OXX XXX XXX XXX, P = OXX XXX XXX XXX.

| Memory Cell | Contents | | Remarks |
|-------------|------------|-------------|-----------------------------------|
| | Left Inst. | Right Inst. | |
| 000 | B 100 | L 014 | |
| 001 | 00 000 | Q 013 | Transfer Control if FE = \$3600 |
| 002 | B 014 | S 100 | |
| 003 | L 102 | T 010 | Transfer Control if \$3600-FE > P |
| 004 | H 099 | B 015 | \$3600-FE → Storage |
| 005 | S 101 | H 098 | \$72-FT → Storage |
| 006 | B 100 | A 099 | } Sum year-to-date FICA earnings |
| 007 | H 100 | B 101 | |
| 008 | A 098 | H 101 | } Sum year-to-date FICA Tax |
| 009 | 50 098 | 90 000 | Print Tax Stop |
| 010 | M 016 | H 098 | .02 X P → Storage |
| 011 | B 102 | H 099 | P → Storage |
| 012 | 00 000 | U 006 | |
| 013 | 50 017 | 90 000 | Print No Tax |
| 014 | 000 000 | 360 000 | |
| 015 | 000 000 | 007 200 | |
| 016 | 002 000 | 000 000 | |
| 017 | NOA FIC | AAI AX. | |

Example Problem #2:

MEDICAL PAY CALCULATION

Memory Cell 100 contains the number of days medical absence, MA, an employee reported. In memory cell 101 is stored his hourly rate of pay, R, while in cell 102 is the number of days remaining of his payable medical absence allowance, ML. Calculate the medical pay, MP, and print, reducing the medical leave allotment by the amount MA.

Assume that the assumed decimal points are:

MA: OXX[^] 000 000 000
 ML: ~~T~~XX[^] 000 000 000
 R : OXX X[^]XX 000 000
 MP: OXX XXX XXX X[^]XX

| Memory Cell | Contents | | | | Remarks |
|-------------|------------|------------|-----|-----|-----------------------------|
| | Left Inst. | Right Inst | | | |
| 000 | B | 100 | | | |
| 001 | 00 | 000 | L | 013 | |
| 002 | B | 102 | Q | 012 | Transfer Control if MA = 0 |
| 003 | 50 | 013 | T | 006 | Transfer Control if ML > 0 |
| 004 | B | 102 | 00 | 000 | } ML-MA → ML |
| 005 | H | 102 | S | 100 | |
| 006 | L | 100 | 90 | 000 | Stop |
| 007 | L | 102 | T | 008 | Transfer Control if ML > MA |
| 008 | M | 014 | 00 | 000 | } 8.rL . R → Storage |
| 009 | L | 099 | H | 099 | |
| 010 | H | 099 | M | 101 | |
| 011 | 00 | 000 | 50 | 099 | Print MP |
| 012 | 50 | 013 | U | 004 | |
| 013 | 000 | 000 | 90 | 000 | Print Zero |
| 014 | 000 | 080 | 000 | 000 | Stop |
| | | | 000 | 000 | |

Student Exercises:

1. Three numbers A, B, and C are stored in memory locations 100, 101, and 102 respectively. Print the smallest and stop.
2. The following data pertaining to an employee is stored in the indicated memory cells:

100: Employee's badge number, BN.
101: Weekly bond deduction, BD.
102: Size of bond to be purchased, BS.
103: Employee's bond account number, BA.
104: Cumulative bond deduction, BC.

Add this week's bond deduction to the cumulative total and initiate purchase of a bond, if appropriate, by printing BN, BS, and BA and make adjustments to the cumulative deductions.

SECTION 1

Introduction

From a logical standpoint Univac instructions (and those of most digital computers) can be grouped into three categories which we might call "logical operations":

- A) Transfer of information from one storage location or medium to another. Typical of the instructions in this category are B m, H m, and L m.
- B) Arithmetic manipulation of information, as typified by the instructions A m, M m, D m.
- C) Decision as to following course A or Course B, based upon the relative magnitudes of numbers. Instructions in this category are the T m, Q m, and redundantly, U m.

"Programming" may then be defined as the act of combining these three logical operations to "solve" a given problem, and a "program" is the resulting combination. "Coding" is the act of translating the program into the instruction code of a particular computer and is sometimes used to designate the completed translation as well.

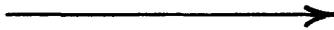
When used in this sense, programming is the most involved step in preparing a problem for a computer solution. Coding the program is very nearly a mechanical process and in several cases computers have been successfully instructed to do their own coding. Construction of the program is the most difficult step for beginning students of digital computer techniques because it often requires visualizing large parts, if not all, of the problem as a unit. To some extent a program will reflect characteristics of the computer at hand. For example, a problem programmed for computer X having six input-output units will differ from the program for computer Y with only three, though often these differences are slight.

Because of the difficulties for the beginner in visualizing correctly all the steps in the problem before coding it, a graphical representation of the problem is often laid out before beginning the coding. This makes it much easier to detect errors in procedure and correct them than if one had to work directly on hundreds of separate instructions. We call this graphical representation a flow chart. Essentially, flow charts consist of symbols for the logical operations we have discussed and rules for assembling and ordering them.

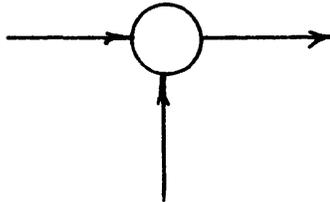
SECTION 11
Flow Chart Symbols

Assembling and Ordering Symbols:

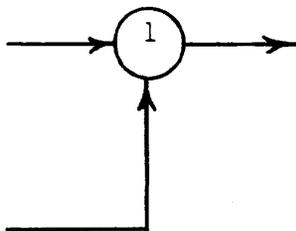
The "path of computational flow" is indicated by a directed line segment:



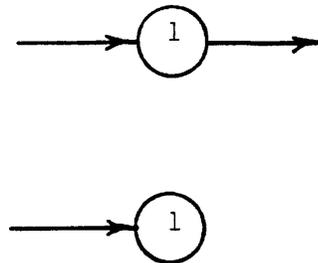
The inference that the next step in the problem will be found by moving along the line in the direction of the arrow is obvious. Where two or more different paths of computational flow merge to follow one common path, a "fixed connector" is placed at the point of merging:



By numbering the fixed connectors we need not indicate a merging of flow paths by the actual joining of the lines as shown above. This is especially advantageous where the merging flow lines would have to come from widely separated areas of the chart. Thus, A and B are identical operations.

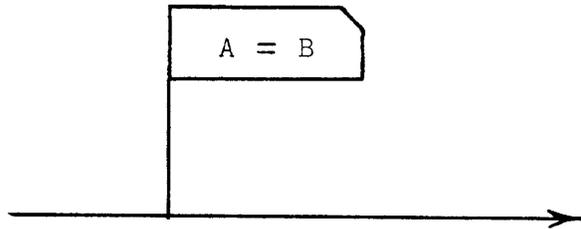


A



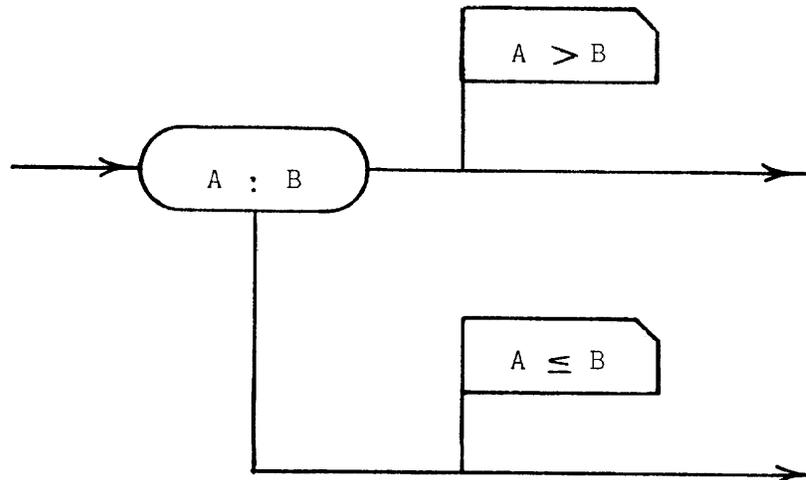
B

If for reason of clarity we wish to indicate that a certain condition is true at a certain point in the line of computational flow, a "flag" asserting the condition is attached to the flow line:

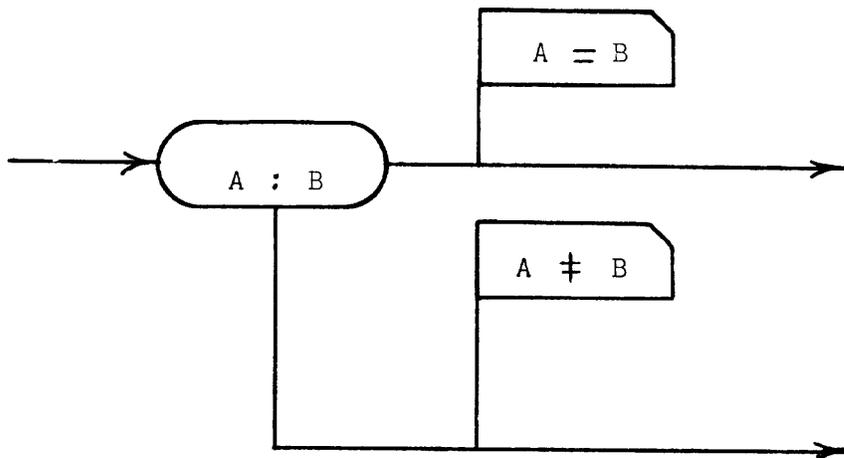


Logical Decision Symbols:

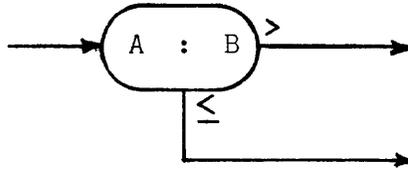
A decision between following one of two paths of computational flow, based upon the relative magnitudes of two quantities A and B is indicated by:



Or, for the choice of paths based upon the equality of the two quantities:



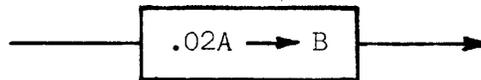
Standard practice has been to dispense with the flags in the case of a decision operation by the following scheme:



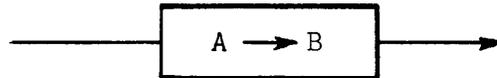
Ambiguity is avoided if we remember that the condition existing on either outward flow line is that obtained by replacing the colon (:) with the $>$ or \leq symbol where the choice is made on relative magnitude or by $=$ or \neq when the choice is based upon equality.

Arithmetic and Transfer Symbols:

The evaluation of a formula or straight computation is indicated inside a rectangular box



Where the arrow indicates that B is now the quantity .02A, a transfer would appear as

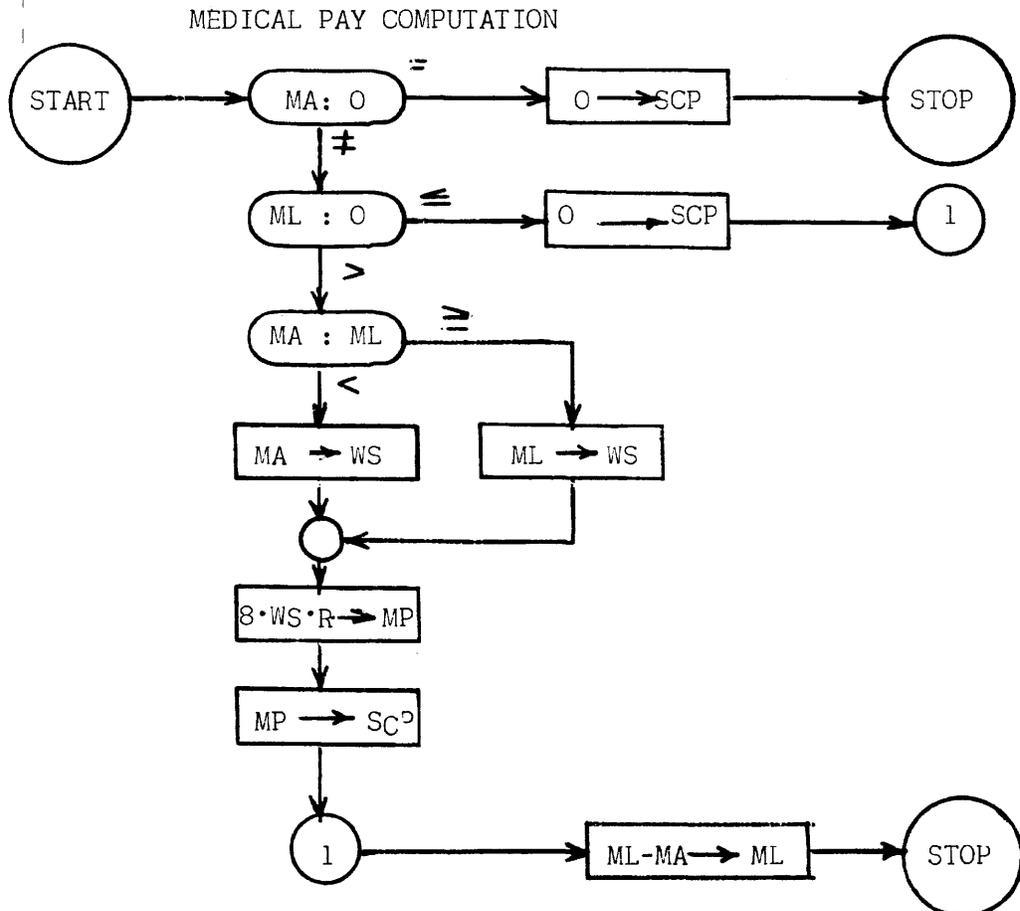
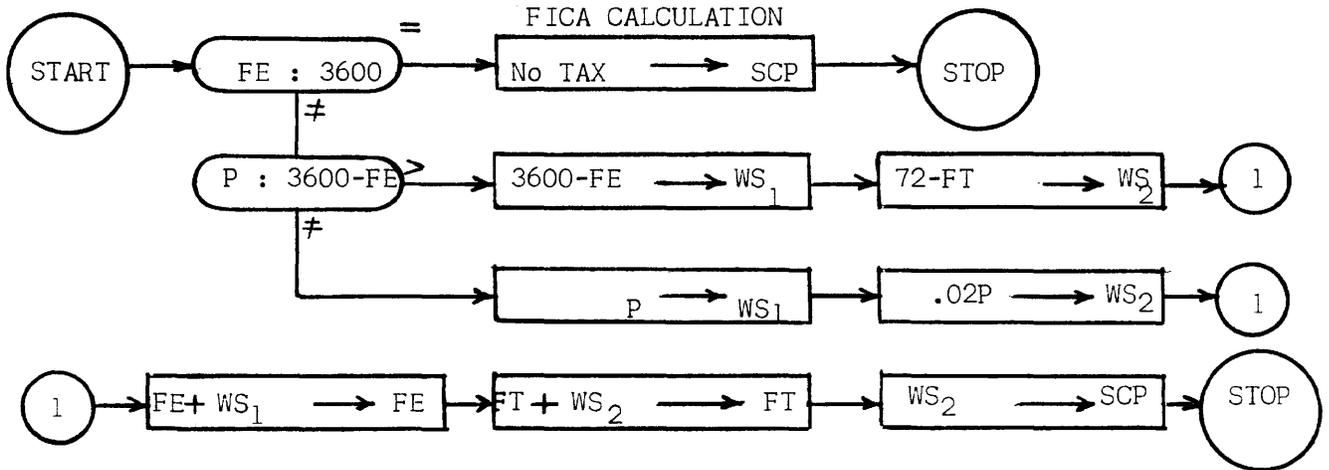


Again the arrow signifies that B is now the Quantity A.

SECTION 111

Example Flow Charts

Let us illustrate the use of these flow chart symbols by drawing the flow charts for the FICA and medical pay calculation problems of Chapter 2.



Student Exercises:

1. Draw the flow charts for the exercises of Chapter 2, Section IV.
2. Flow chart and code the following billing problem:

in memory cells 100 - 104 are certain order and billing information:

100: Quantity ordered, Q: 0XX XX[^]0 000 000
101: Unit Price, P: 0XX XXX[^] XX0 000
102: Percentage Discount for:
quantities over 50, D: 0[^]XX X00 000 000
103: Salesman's number, N
104: Commission percentage, C: 0[^]XX X00 000 000

Compute the commission (on net charge), print with salesman's number.

SECTION 1

Iterative Coding

Consider the following problem:

An account number, A, is stored in memory cell 099 while in memory cells 100-199 is stored a list of 100 delinquent account numbers. Print "No Credit" if A is in the delinquent list and "Credit Good" if it is not.

Let us write down the coding necessary to determine whether A is the same as the first delinquent account:

```

000  B    100
001  00   000      L    099
002  -    -        Q    007      Transfer Control if A = First DA
003  -    -        -
004  -    -        -
005  -    -        -
006  -    -        -
007  50   008      90   000      Print "No Credit"
008  NOA  CRE      DIT.▲▲      Stop
009  -    -        -
010  -    -        -
011  -    -        -

```

Thus, if the computer gets to line 002 we know that the first delinquent account is not the same as A. Now, let us write the coding necessary to compare the second delinquent account with A:

```

000  B    101
001  00   000      L    099
002  -    -        Q    007      Transfer Control if A = Second DA
003  -    -        -
004  -    -        -
005  -    -        -

```

| | | | | |
|-----|-----|-----|-----|-----|
| 006 | - | | -- | |
| 007 | 50 | 008 | | |
| 008 | NOA | CRE | 90 | 000 |
| | | | DIT | .AA |
| | | | DIT | |
| 009 | - | | | |
| 010 | - | | | |
| 011 | - | | | |

Print "No Credit"
Stop

And for the third:

| | | | | |
|-----|-----|-----|-----|-----|
| 000 | B | 102 | | |
| | | | L | 099 |
| 001 | 00 | 000 | | |
| | | | Q | 007 |
| 002 | - | | | |
| 003 | - | | | |
| 004 | - | | | |
| 005 | - | | | |
| 006 | - | | | |
| 007 | 50 | 008 | | |
| | | | 90 | 000 |
| 008 | NOA | CRE | | |
| | | | DIT | .AA |
| 009 | - | | | |
| 010 | - | | | |
| 011 | - | | | |

Transfer control if A = Third DA

Print "No Credit"
Stop

And for the hundreth:

| | | | | |
|-----|----|-----|----|-----|
| 000 | B | 199 | | |
| | | | L | 099 |
| 001 | 00 | 000 | | |
| | | | Q | 007 |
| 002 | - | | | |
| 003 | - | | | |
| 004 | - | | | |
| 005 | - | | | |
| 006 | - | | | |
| 007 | 50 | 008 | | |
| | | | 90 | 000 |

Transfer Control if A = Last DA

Print "No Credit"
Stop

| | | | | |
|-----|-------------|-----|-----|----------------|
| 008 | NO Δ | CRE | | |
| | | | DIT | $\Delta\Delta$ |
| 009 | - | | | |
| | | | - | |
| 010 | - | | | |
| | | | - | |
| 011 | - | | | |
| | | | - | |

In each case you will note that the only change is in line 000, the address of the delinquent account being advanced by one.

It is hardly a labor-saving or even feasible scheme to write the 100 tests necessary to ascertain whether A is among the bad account list. The fact, that instructions and data are both stored in the same memory allows us to alter the coding for testing the first account so that this same coding can be repeated, but this time testing the second account, etc.

To do this, we write down, as before, the coding necessary to examine the first delinquent account which is stored in cell 100. Then, by addition, we alter the address of those instructions referring to address 100:

| | | | | | |
|-----|-------------|-----|-----|----------------|----------------------------------|
| 000 | B | 100 | | | |
| | | | L | 099 | |
| 001 | 00 | 000 | | | |
| | | | Q | 007 | Transfer control if A = First DA |
| 002 | B | 000 | | | |
| | | | - | | |
| 003 | - | | | | |
| | | | - | | |
| 004 | A | 009 | | | |
| | | | H | 000 | Augment left address of |
| 005 | 00 | 000 | | | line 000 by 1 |
| | | | U | 000 | |
| 006 | - | | | | |
| | | | - | | |
| 007 | 50 | 008 | | | |
| | | | 90 | 000 | Print "No Credit" |
| 008 | NO Δ | CRE | | | Stop |
| | | | DIT | $\Delta\Delta$ | |
| 009 | 000 | 001 | | | |
| | | | 000 | 000 | |
| 010 | - | | | | |
| | | | - | | |
| 011 | - | | | | |
| | | | - | | |

The computer will begin by testing the first delinquent account. If it is not A, it will alter its instructions so that it can examine the second DA, etc., until, and if, it finds a DA = A whence it prints "No Credit" and stops. Of course, if A is not on the delinquent list, there is nothing so far to instruct the machine to print "Credit Good" and stop. We wish this process to stop when we have examined the last delinquent account (in memory cell 199) and have found it not equal to A. This can be accomplished by examining the changing instruction line at each iteration:

| | | | | |
|-----|-------------|-----|-------------|----------|
| 000 | B | 100 | L | 099 |
| 001 | 00 | 000 | Q | 007 |
| 002 | B | 000 | L | 010 |
| 003 | 00 | 000 | Q | 006 |
| 004 | A | 009 | H | 000 |
| 005 | 00 | 000 | U | 000 |
| 006 | 50 | 011 | 90 | 000 |
| 007 | 50 | 008 | 90 | 000 |
| 008 | NO Δ | CRE | DIT | Δ |
| 009 | 000 | 001 | 000 | 000 |
| 010 | B00 | 199 | L00 | 099 |
| 011 | CRE | DIT | Δ GO | OD. |

Transfer control if first DA = A

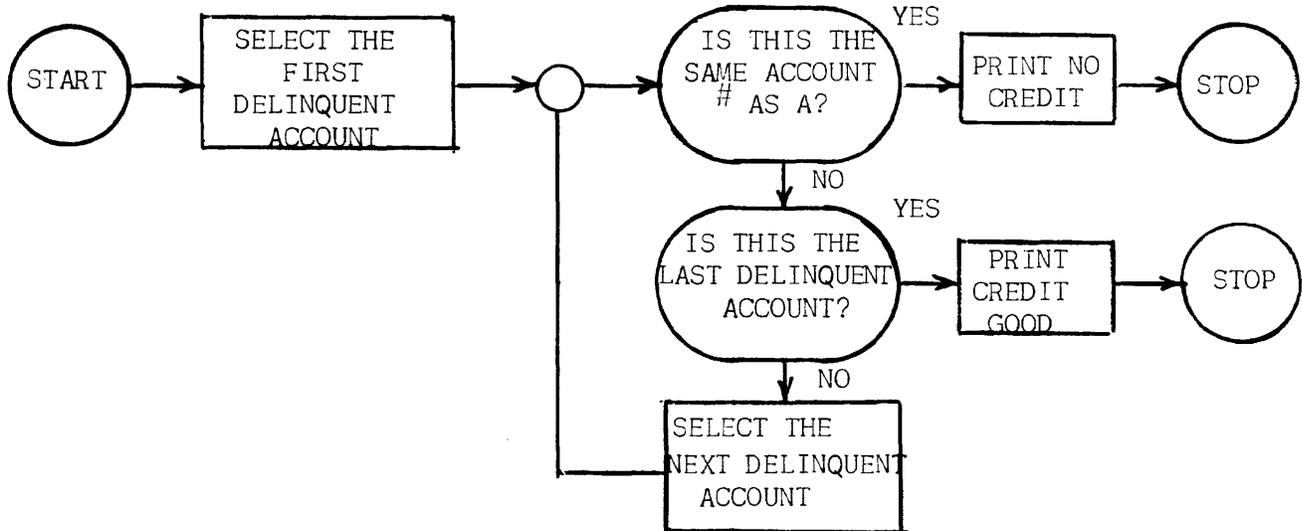
Transfer control if last DA
 Augment left address of
 line 000 by 1

Print "Credit Good"
 Stop
 Print "No Credit"
 Stop

SECTION 11

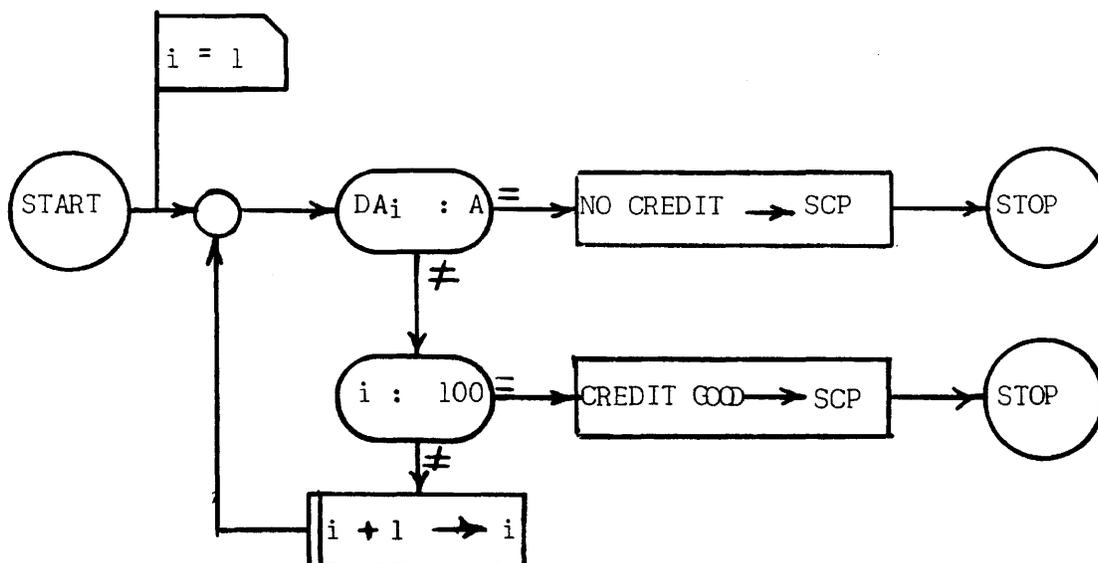
Subscript Notation And
Additional Flow Chart Symbols

Let us draw a word picture of this process so as to fix more firmly in our minds the essential steps of the solution:



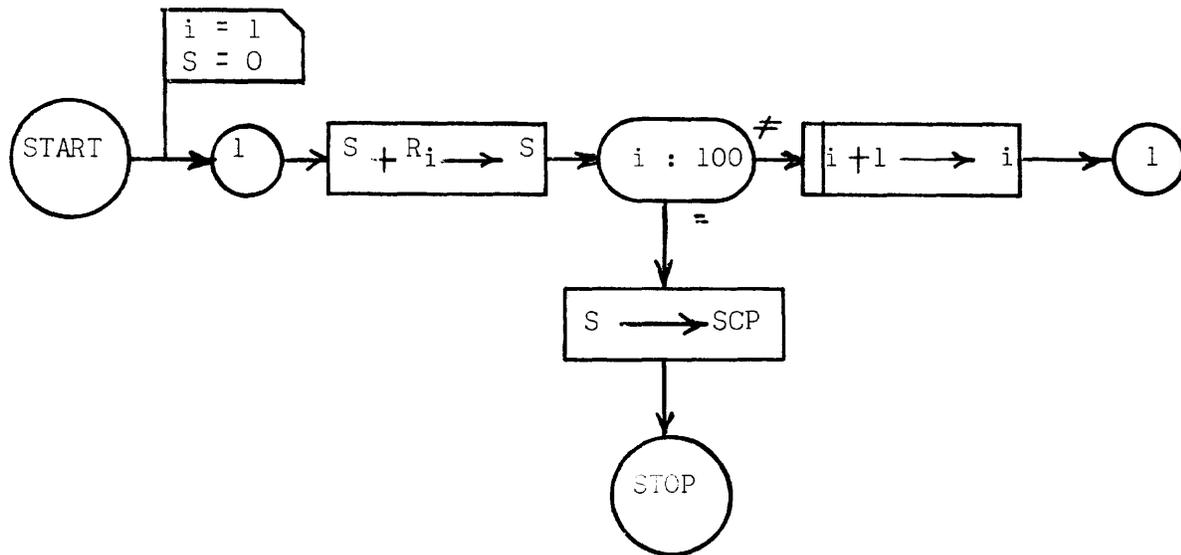
The word flow chart is a fair description of this problem. Realistic business problems, however, would require reams of paper for word charts and even at best the word chart can be confusing.

In order to draw a more economical chart, we need an unambiguous way of designating a particular delinquent account. A commonly accepted method is through the use of subscripts. Let us call the first delinquent account DA_1 the second delinquent account DA_2 and so forth, the last being DA_{100} . Now the symbol DA_i can be read as the i th delinquent account, i being some number between 1 and 100. Our flow chart will now appear as:



The double-lined box $i + 1 \rightarrow i$ is used when altering subscripts to remind us that we are changing instructions rather than data. Another simple example of the iterative coding technique is the following problem:

A list of 100 receipts, R_i , are stored in memory cells 100 to 199. Compute and print their sum.



| | | | | | |
|-----|-----|-----|-----|-----|--|
| 000 | B | 006 | | | } $S + R_i \rightarrow S$ |
| 001 | H | 006 | A | 100 | |
| 002 | L | 007 | B | 000 | } Transfer control if $i = 100$ $i + 1 \rightarrow i$ |
| 003 | A | 008 | Q | 005 | |
| 004 | 00 | 000 | H | 000 | |
| 005 | 50 | 006 | U | 000 | } Print S Stop S |
| 006 | 000 | 000 | 90 | 000 | |
| 007 | B00 | 006 | 000 | 000 | |
| 008 | 000 | 000 | A00 | 199 | |
| | | | 000 | 001 | |

Student Exercises for Flow Charting and Coding:

1. In memory cells 100-199 are a series of 100 inventory amounts, M_i , while in cells 200-299 are a series of corresponding sales, S_i . That is, S_1 and M_1 refer to the same stock item, S_2 and M_2 , etc. Correct the inventory.
2. Do the second example in the text, where now the number of receipts is unknown. A "Sentinel word" consisting of ZZZ ZZZ ZZZ ZZZ will follow the last receipt.

3. A net pay, P, after taxes, is stored in memory cell 099. In memory cells 100.... are stored a series of deductions, D_i . Their number is not known, but a Z sentinel follows the last one. (note: there may be no deductions, in which case the sentinel is in cell 100.) Each deduction is to be examined in the following manner: If the deduction will not reduce the pay below \$15 it is to be subtracted from the pay, producing a new net pay. If however, this deduction will bring the pay below \$15 do not subtract it from the pay. Either the entire deduction is to be taken from the pay or none of it, that is, no partial deduction will be made. The net pay is to be printed along with all deductions that have not been made.

SECTION 111

Instruction List C

The instructions in list C permit us to alter the arrangement of the characters in a word in an arbitrary manner. The alteration always takes place in Register A.

On 000, This is the left shift instruction. The second instruction digit, $n = 1, 2, \dots, 9$, tells the computer to shift all digits of rA, except digit position 1 (the sign) n places left, putting n zeros in the vacated positions on the right. For example, if rA contains 012 845 ABC 968 and the instruction 04 000 is given, rA will then contain 05A BC9 680 000.

-n 000, This is the right shift instruction. The second instruction digit, $n = 1, 2, \dots, 9$, tells the computer to shift all digits of rA, except digit position 1 (the sign digit) n places to the right, putting zeros in the vacated positions on the left. For example, if rA contains 012 345 ABC 968 and the instruction -4 000 is given, rA will then contain 000 001 234 5AB.

To illustrate the use of these instructions let us consider the table look-up problem:

A shipping rate table of 100 entries is stored in memory cells 100-199. The rate in cell 100 is the cost/pound to be levied for all items of weight 0 to 99 pounds. The rate in cell 101 applies to 100-199 pound items, etc., in memory cell 099 is the weight of an item to be shipped, within the range of the table, in this format:

000 XXX X[^]00 000

Thus, a weight of 3523 pounds would be stored in 099 as 000 352 300 000. The rates are given in dollars and cents/pound in the following format (the example is \$2.36/lb.):

000 2[^]36 000 000

Compute and print the shipping charge.

In solving this problem, which occurs in a variety of forms in data processing, let us note that the hundreds and thousands digits of (099) are in one-to-one correspondence with the memory addresses of their corresponding rates. For example, if the weight is between 2100 & 2199 lbs. The appropriate rate is in cell 121. Thus, we can use the hundreds and thousands digits of the weight to locate the applicable rate in the table. The coding for this problem is given below:

| | | | | | |
|-----|----|-----|----|-----|-----------------------|
| 000 | B | 099 | | | 000 XXX X 00 000 → rA |
| | | | -7 | 000 | 000 000 000 OXX → rA |
| 001 | A | 005 | | | L00 099 M00 LXX → rA |
| | | | H | 002 | rA → 002 |
| 002 | - | | | | |
| | | | - | | Rate x Weight → rA |
| 003 | H | 098 | | | |
| | | | 50 | 098 | Print Shipping Cost |
| 004 | 90 | 000 | | | Stop |
| | | | - | | |
| 005 | L | 099 | | | |
| | | | M | 100 | |

A simple modification allows us to do table look-up problems when the table arguments are not multiples of 100. For example, suppose these same rates applied to shipping weights in intervals of 250 pounds. Now, if we multiply the weight by 1/250 (= .004), we have the number of 250 pound units, which is in one-to-one correspondence with the memory address of the rates.

| | | | | | |
|-----|-----|-----|-----|-----|------------------------------------|
| 000 | L | 099 | | | 00X XXX X ^A 00 000 → rA |
| | | | M | 006 | Weight/250 = 000 00X X 00 000 → rA |
| 001 | -5 | 000 | | | 000 000 000 OXX → rA |
| | | | A | 007 | L00 099 M00 LXX → rA |
| 002 | H | 003 | | | rA → 003 |
| | | | 00 | 000 | |
| 003 | - | | | | |
| | | | - | | Rate x Weight → rA |
| 004 | H | 098 | | | |
| | | | 50 | 098 | Print Shipping Cost |
| 005 | 90 | 000 | | | Stop |
| | | | - | | |
| 006 | 000 | 400 | | | 1/250 |
| | | | 000 | 000 | |
| 007 | L00 | 099 | | | |
| | | | M00 | 100 | |

SECTION IV

Instruction List C (continued)

F m. This instruction duplicates in rF, which is a one word register, the contents of memory cell m.

E m. This is the extract instruction. The contents of rA are selectively replaced by the contents of m. The word in rF, called the extractor, governs the transfer. Of the 63 Univac characters, every other one, starting with the ignore (i) extracts, in particular, the l. Thus, if the initial contents of rA, rF, and m are:

```

m:   123 ABC 789 DOE
rF:  010 110 001 101
rA:  987 6DE 017 54G
    
```

And the instruction E m is given, the contents of rA will be:

```

rA:  927 ABE 019 D4E
    
```

Two examples will illustrate some of the uses for this instruction.

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 | L | 100 | | 000 XXX XXX XXX → rL |
| | | | B | 006 001 --- --- --- → rA |
| 001 | -1 | 000 | | Shift rA right one place |
| | | | T | 001 Transfer control if rA > rL |
| 002 | H | 008 | | } rA → rF |
| | | | F | |
| 003 | B | 007 | | |
| | | | E | 100 MM MM MM MM → rA |
| 004 | H | 008 | | |
| | | | 50 | 008 Print edited quantity |
| 005 | 90 | 000 | | Stop; |
| | | | 00 | 000 |
| 006 | 001 | --- | | |
| | | | -- | --- |
| 007 | MM | MM | | |
| | | | MM | MM |
| 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.

Suppose, now, that the word in 100 is in the following format:

100: 000 XXX XX[^]X XXX

where[^] indicates the position of the assumed decimal point. We wish to print this number with decimal point, and nonsignificant zeros suppressed. That is, if 100: 000 003 607 690 we will print 360.7690. The first step is to "spread" the word apart and insert the decimal point, then zero suppression.

| | | | | | |
|-----|-----|-----|-----|-----|------------------------------------|
| 000 | B | 100 | | | 000 XXX XX [^] X XXX → rA |
| | | | 01 | 000 | 00X XXX X [^] XX XX0 → rA |
| 001 | H | 014 | | | rA → Working Storage |
| | | | -1 | 000 | 000 XXX XXX XXX → rA |
| 002 | A | 010 | | | 000 XXX X.X XXX → rA |
| | | | F | 011 | 001 111 100 000 → rF |
| 003 | E | 014 | | | 00X XXX X.X XXX → rA |
| | | | H | 014 | rA → Working Storage |
| 004 | L | 014 | | | |
| | | | B | 012 | |
| 005 | -1 | 000 | | | |
| | | | T | 005 | |
| 006 | H | 015 | | | Zero Suppression |
| | | | F | 015 | |
| 007 | B | 013 | | | |
| | | | E | 014 | |
| 008 | H | 014 | | | |
| | | | 50 | 014 | |
| 009 | 90 | 000 | | | |
| | | | 00 | 000 | |
| 010 | 000 | 000 | | | |
| | | | 0.0 | 000 | |
| 011 | 001 | 111 | | | |
| | | | 100 | 000 | |
| 012 | 01- | --- | | | |
| | | | --- | --- | |
| 013 | △△△ | △△△ | | | |
| | | | △△△ | △△△ | |
| 014 | - | | | | |
| | | | | | |
| 015 | - | | | | |

} Working Storage

SECTION 1

Introduction

Most of our study of computer applications so far has concerned itself with single units of information. In a few problems we saw that the processing required several bits of data before the computation could begin. This leads us to the concept of the item. An item is a set of related data, called fields, which may be treated as a unit. The composition of an item (that is the fields of which it is composed) may depend on the processing to be done, the policies of a company, or the manner in which the data is obtained. Bearing these reasons for variation in mind, examples of typical items are given below:

A Master Employee Item

- 1) Name and Address
- 2) Badge or Employee Number
- 3) Rate of Pay
- 4) Number of Dependents
- 5) Type of Standard Deductions
- 6) Work and Pay Summaries

A Stock Inventory Item

- 1) Stock Number
- 2) Description
- 3) Unit of Measure
- 4) Lead Time for Ordering
- 5) Unit Cost
- 6) Amounts on Hand and on Order

A Public Utility Meter Item

- 1) Account Number
- 2) Current Reading
- 3) Last Reading

In the Univac, an item may consist of one or more words. Certain considerations should be observed when laying out the item:

- 1) Do not split a field across words if it can be avoided.
- 2) Fields which specify an order or sequence for the items should be in the same word, beginning at the extreme left and continuing to the right in decreasing order of significance.
- 3) Fields upon which considerable calculation will be performed should be in separate words if possible.

- 4) Keep item sizes as small as feasible, bearing in mind the number of instructions required to "move" the item about and the possibility of future expansion.

The stock inventory and utility meter items are pictured below to illustrate the appearance of an item.

The item notation used throughout this course is a simple one and yet is flexible and unambiguous. A letter will stand for the set of all items of a particular kind, a subscript defining a specific item of the set. A superscript will serve to identify the field under consideration. For example, let S stand for all stock inventory items, then S_i is the i th such item, while

S_i^{sn} is the Stock number of the i th inventory item
 S_i^d " " Description " " " " "
 S_i^u " " Unit of Measure of the i th inventory item

Etc

Stock Inventory Item

| | | | | | | | | | |
|--------------------|-----------|--|--|--|---|---|-----------------|--|--|
| Stock Number | | | | | | | | | |
| Description | | | | | | | | | |
| Description (Cont) | | | | | | | Unit of Measure | | |
| 0 | On Hand | | | | | | | | |
| 0 | On Order | | | | | | | | |
| 0 | Unit Cost | | | | 0 | 0 | Lead Time | | |

^

Public Utility Meter Item

| | | | | | | | | | |
|-----------------|--|--|--|---|--------------|---|---|---|---|
| Account Number | | | | 0 | 0 | 0 | 0 | 0 | 0 |
| Current Reading | | | | | Last Reading | | | | |

SECTION 11

Instruction List D

To facilitate the movement of items inside the computer, two multiple word transfer registers have been provided.

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 |

SECTION III

Working Storage

The following problem will illustrate the multiple word transfer instructions and the most efficient means of item handling.

6 ten word items are stored in memory cells 100-159.
Then item layout is:

| |
|----------------|
| Job Number |
| Contract Price |
| Labor Cost |
| Material Cost |
| Overhead Cost |
| |
| OTHER |
| DATA |
| |
| |
| |

Compute the net profit and store the job number and profit as a two-word item in cells 160-171.

We shall solve this problem in two ways. In the first let us code the steps necessary to process the first item (100-109) and store the desired fields in 160-161 and then process the remaining items by appropriately increasing the address part of the instructions referring to those locations:

```

000   B   100           ) Job Number → 160
           H   160       )
001   B   101           )
           S   102       )
002   S   103           ) Net Profit → 161
           S   104       )
003   H   161           )
           B   000       )
004   L   012           )
           Q   011       ) Transfer Control if last item
005   A   013           )
           H   000       )
006   B   001           )
           A   014       )
007   H   001           )
           B   002       ) Augment Addresses
008   A   014           )
           H   002       )
009   B   003           )
           A   015       )
010   H   003           )
           U   000       )

```

| | | | | | |
|-----|-----|-----|-----|-----|------|
| 011 | 90 | 000 | | | Stop |
| | | | 00 | 000 | |
| 012 | B | 150 | | | |
| | | | H | 170 | |
| 013 | 000 | 010 | | | |
| | | | 000 | 002 | |
| 014 | 000 | 010 | | | |
| | | | 000 | 010 | |
| 015 | 000 | 002 | | | |
| | | | 000 | 000 | |

Now, if we recode the problem, this time instead of increasing all instructions, which refer to addresses 100-109, we simply transfer the next input item into the position occupied by the first item.

| | | | | | |
|-----|-----|-----|-----|-----|----------------------------------|
| 000 | B | 101 | | | } Net Profit → 101 |
| | | | S | 102 | |
| 001 | S | 103 | | | } Store Computed Item |
| | | | S | 104 | |
| 002 | H | 101 | | | } Replace 100-109 with next item |
| | | | V | 100 | |
| 003 | W | 160 | | | } Transfer if last item computed |
| | | | Y | 110 | |
| 004 | Z | 100 | | | } Stop |
| | | | B | 003 | |
| 005 | L | 009 | | | |
| | | | Q | 008 | |
| 006 | A | 010 | | | |
| | | | H | 003 | |
| 007 | 00 | 000 | | | |
| | | | U | 000 | |
| 008 | 90 | 000 | | | |
| | | | 00 | 000 | |
| 009 | W | 170 | | | |
| | | | Y | 160 | |
| 010 | 000 | 002 | | | |
| | | | 000 | 010 | |

Observe that this technique uses considerably fewer steps. The point to be made is that when a problem requires many items to be put through a process the efficient method is to code the routine specifically addressed for the first input item; then, considering the location of this item as working storage, transfer the remaining items successively into this position for processing.

Student Exercise

1. Memory locations 100-159 contain a series of two word census items:

OSS 000 CCC C00
OAA AOM OH0 00G

Where SS is a two digit state code
CCCC " " four " city "
AAA " the individual's age
M " " " marital status

S = Single
M = Married
W = Widowed
D = Divorced

H is the individual's race
W = White
C = Colored

G is the sex
M = Male
W = Female

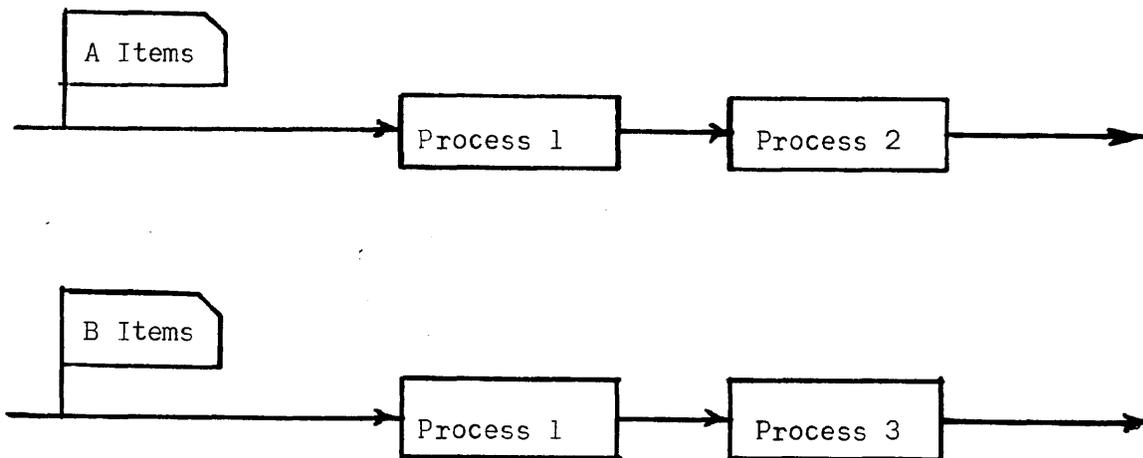
Print the total number of white, single, females, 21 years of age or older in Sheboygan (city code 1313), Wisconsin (state code 24).

SECTION IV

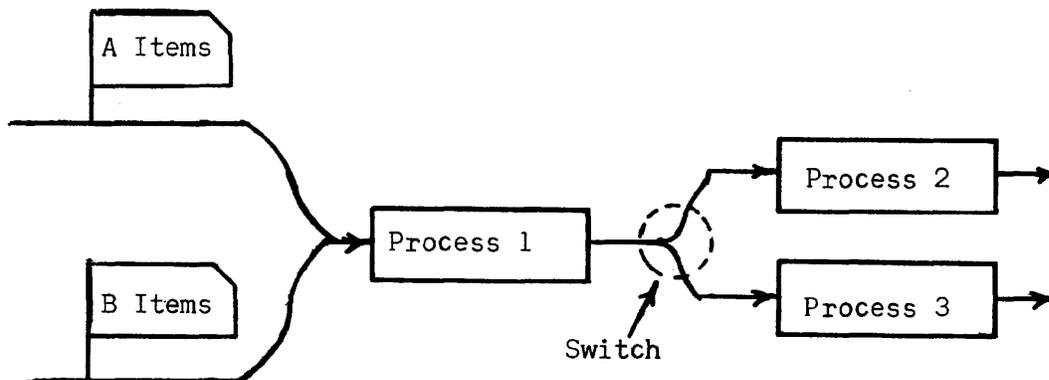
Variable Connectors

You recall that the fixed connector was used on a computational flow line to indicate a merging of flow lines. Its counterpart, the variable connector, is used where flow-lines diverge.

Suppose we have a problem to code requiring us to send Type A items through process 1 and then through process 2, while Type B items must go through process 1 then process 3. The normal procedure would be to repeat process 1 along the A flow line and B flow line:



If process 1 is a complex and extensive calculation it becomes highly inefficient to repeat it along every flow line requiring it. The suggestion immediately presents itself of doing process 1 once only and funneling both items through it, separating them again after completion of process 1:



When used in this manner, process 1 is called a Subroutine. This flow chart leaves an ambiguous situation at the diverging flow lines, do "A" items go to process 2 or 3, and how is the switch actually accomplished?

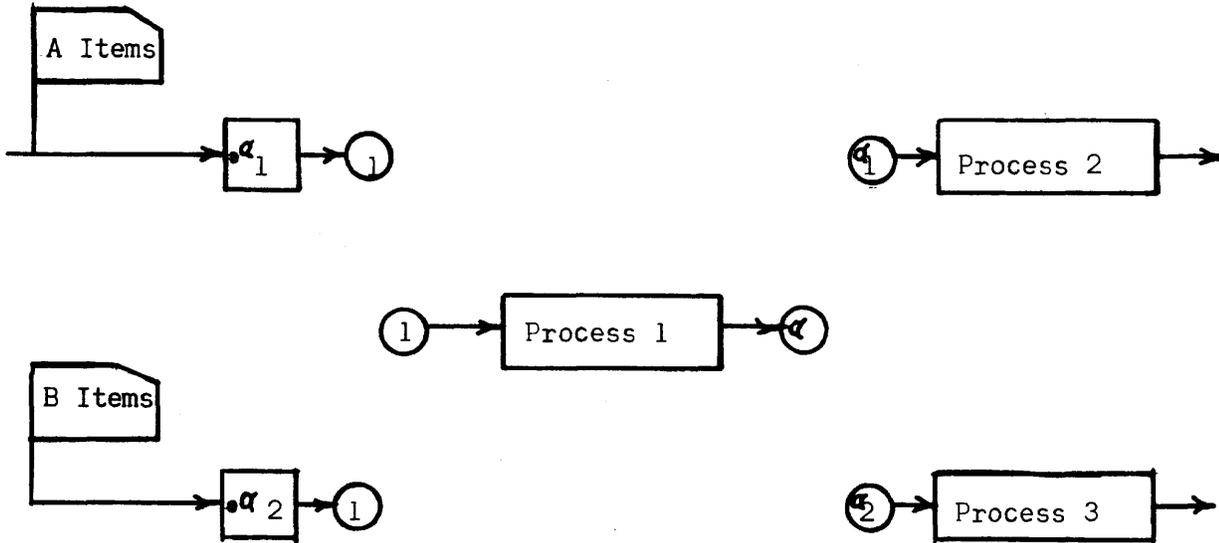
The switch is called a "Variable Connector" and is indicated by:



The letters of the Greek alphabet are used to designate the variable connector. The act of setting the connector is shown by the symbol.



Which implies that, after passing through the box, when we encounter the connector α it is set to read α_1 . Our example would now appear:

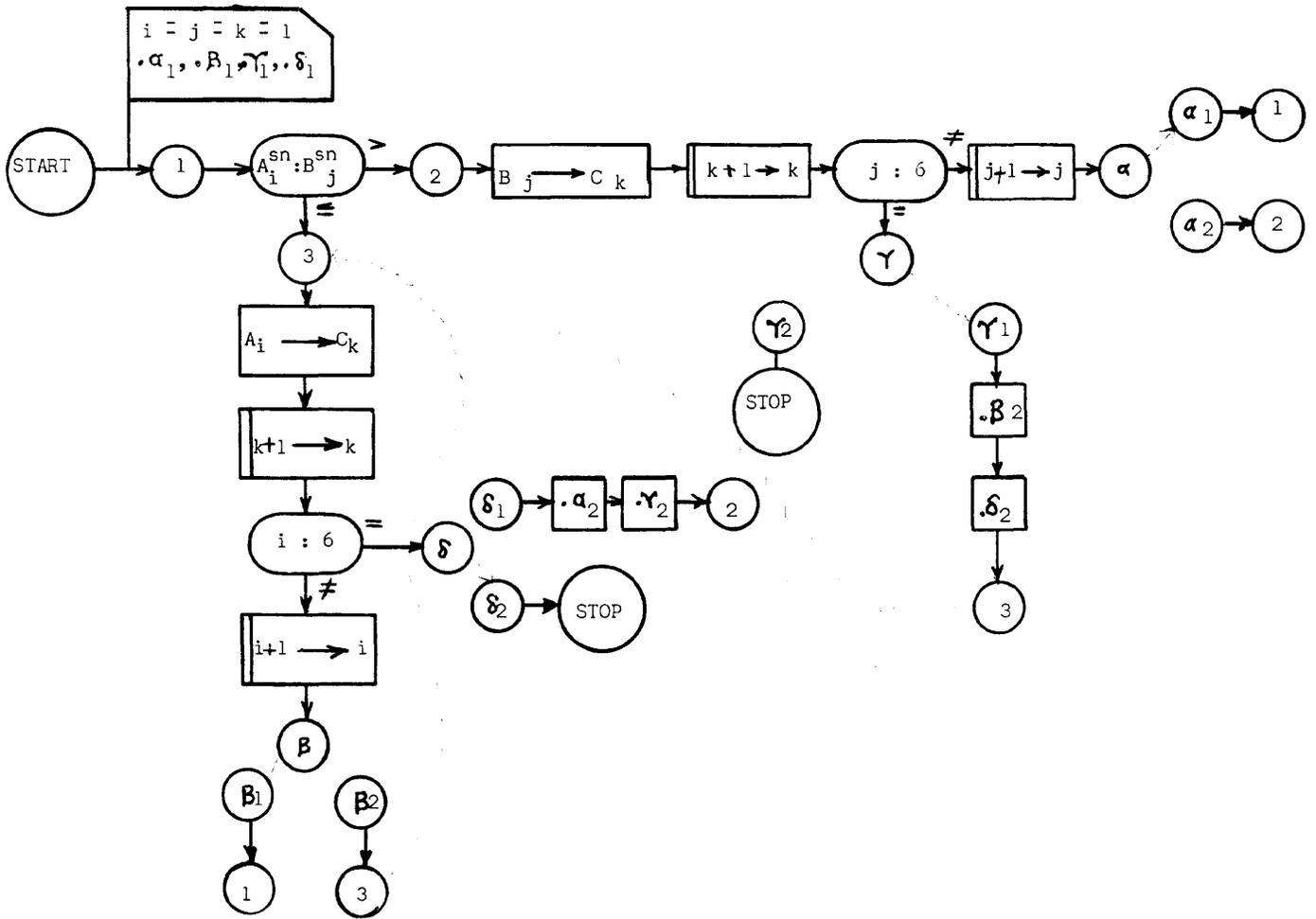


This makes evident that a variable connector is the same as a fixed connector, but the number the connector transfers us to is determined when the connector is "set".

To illustrate the use of these connectors, consider the following problem:

6 ten-word "A" items are stored in memory cells 100-159. Each item has a serial number as its first word and the items are in ascending order by this serial number. Similarly, 6 ten-word "B" items are stored in 200-259. Merge (interfile) these items together, storing the merged results in 300-419.

In the flow chart and coding A_i represents the i th A item, B_j the j th B item and C_k the k th output item. The superscript SN stands for the serial number of the indicated item.



| | | | | |
|-----|----|-----|---|-----|
| 000 | B | 100 | L | 200 |
| 001 | 00 | 000 | T | 012 |
| 002 | Y | 100 | Z | 300 |
| 003 | B | 002 | A | 023 |
| 004 | H | 002 | B | 012 |
| 005 | A | 023 | H | 012 |
| 006 | B | 009 | L | 024 |
| 007 | 00 | 000 | Q | 011 |
| 008 | A | 025 | H | 009 |
| 009 | Y | 100 | Z | 100 |
| 010 | 00 | 000 | U | 000 |
| 011 | V | 026 | W | 020 |
| 012 | Y | 200 | Z | 300 |
| 013 | B | 012 | A | 023 |
| 014 | H | 012 | B | 002 |
| 015 | A | 023 | H | 002 |
| 016 | B | 019 | L | 030 |
| 017 | 00 | 000 | Q | 021 |
| 018 | A | 025 | H | 019 |
| 019 | Y | 200 | Z | 200 |
| 020 | 00 | 000 | U | 000 |
| 021 | V | 028 | W | 010 |
| 022 | 00 | 000 | U | 002 |

Transfer Control if $A_i^{sn} > B_j^{sn}$
 $A_i \rightarrow C_k$

$k + 1 \rightarrow k$

Transfer Control if $i = 6$

$i + 1 \rightarrow i$

Variable Connector β

Variable Connector δ (α_2 and γ_2)

$\left. \begin{array}{l} B_j \rightarrow C_k \\ k + 1 \rightarrow k \end{array} \right\}$

Transfer Control if $j = 6$

$j + 1 \rightarrow j$

Variable Connector α

Variable Connector γ
 β_2 and γ_2

| | | | | |
|-----|-----|-----|-----|-----|
| 023 | 000 | 000 | 000 | 010 |
| 024 | Y | 150 | Z | 100 |
| 025 | 000 | 010 | 000 | 000 |
| 026 | 00 | 000 | U | 012 |
| 027 | 90 | 000 | 00 | 000 |
| 028 | 00 | 000 | U | 002 |
| 029 | 90 | 000 | 00 | 000 |
| 030 | Y | 250 | Z | 200 |

SECTION 1

Characteristics of Magnet Tape

As mentioned in Chapter 1, magnetic tape forms the principal means of introducing and removing information from the Univac. Physically, the tape is a metallic strip about 1/2 inch wide and .002 inches thick. It can be supplied in reels of various sizes, the longest being about 1500 feet. Metallic tape is used because of its excellent resistance to moisture variations and fire, as compared with paper or plastic tapes. Certified tests have shown the tape to be readable after direct contact with temperatures as high as 550°F. An outstanding characteristic of metallic tapes is their reuseability. Information may be recorded on a tape, read, erased and new information recorded on this same tape, thus cutting the cost of supplies considerably. A reel of tape can be read or written upon nearly 1000 times before showing appreciable wear.

Univac characters are recorded serially along the length of the tape at varying densities, up to a high density of 128 characters/inch. A full reel of tape, 1500 feet, can hold as much as 1,440,000 digits, in a cubic space of less than 40 inches, providing a compact information storage medium, which is nearly age proof, without the necessity of moisture or air conditioning. Information recorded on tape is in block form, 720 digits or 60 words to a block. Block recording is used to permit very high reading and recording speeds, a necessity in data processing problems. The 60 word block was chosen as a programming convenience. This block size permits the greatest variation of item size which still retains an integral number of items per block. Thus, a block may consist of

| | | | | | | | |
|----|---|------|-------|---|----|------|-------|
| 60 | 1 | word | items | 6 | 10 | word | items |
| 30 | 2 | " | " | 5 | 12 | " | " |
| 20 | 3 | " | " | 4 | 15 | " | " |
| 15 | 4 | " | " | 3 | 20 | " | " |
| 12 | 5 | " | " | 2 | 30 | " | " |
| 10 | 6 | " | " | 1 | 60 | " | " |

SECTION 11

Recording on Tape

Three means, at present, are available for recording on tape:

Key Board to Tape Recording:

Unityper 1 is a device for converting a typist's key stroke onto magnetic tape. This device contains automatic checking features on the typist. By properly preparing punched paper tape loops, a forced check can be made, preventing the typist from under- or over- typing a field. These loops also provide for automatically filling complete or partial fields with zero, space, or ignore symbols. By operation of an erase key the typist can backspace the tape and re-record if she detects a typing error. The output of Unityper 1 is magnetic tape recorded at a density of 20 characters per inch. A Uniprinter may be plugged directly to the typer to obtain a printed record of the typing (see Uniprinter).

Unityper 11 is a smaller and simpler keyboard-to-tape recorder suitable for volume data recording. It consists of a slightly modified Remington Electric typewriter with an integrally attached tape mechanism and power supply and can be mounted on the normal typist desk. As each key is struck, the character is printed and also recorded on tape. Similar erasing and fill-in features are provided as in the Model I Unityper, but no loops for forced check on field size have been included. Each line of print (120 characters) is recorded on tape as a ten word item. The recording is at a density of 50 characters/inch.

Card To Tape Recording:

The Card-To-Tape Converter, as its name implies, is a device for converting the information on 80 column punched cards to magnetic tape. Each card is converted to a ten word item (called a blockette) on tape. The conversion is governed by a detachable plugboard. Thus, the columns on the card may be plugged to any desired digit position of the 120 character blockette. Unplugged columns and the remaining 40 columns of the blockette may be filled with zero or space symbols. Provision is made for separation of the over punches in a column and recording a column as two separate digits. This is a checked conversion, a card is ready, stored in a memory unit and then recorded on tape. Next, the tape is backspaced to the beginning of the blockette, the card is then reread at a different reading station. The information going to the same memory, but the memory cells and circuitry are scrambled. During the second reading of the card the blockette already recorded on tape, is read and a comparison made between the recording and the second storage of the card. Any comparison failure causes the converter to stop. Failure to feed a card is caught by causing each card fed to generate the impulse to feed the next card. Conversion takes place at a rate of 240 cards/minute. The tape is recorded at a character density of 128 per inch, but due to the spacing required between blockettes on tape, a full reel of tape will hold the information on 5000 cards.

Computer To Tape Recording:

This is accomplished by the Uniservos which are described in Section 1V.

SECTION 111

Reading Tape

Tape To Printed Copy:

The High Speed, or line printer is a device for large volume printing of the data recorded on magnetic tape. Each ten-word item recorded on tape at 128 characters/inch printed on a 120 character line, or, if desired, through format plugboards, selected fields of the item may be printed on as many as six different lines. Through use of these same plugboards the digits in the 10-word item may be printed in any desired column, and provision is also made for automatically suppressing nonsignificant zeros. A punched paper tape loop controls the vertical format of the printing. The present model uses continuous form stock, although a cut form attachment is in development. The rate of printing will of course depend on the vertical format to some extent, but speeds of 300 to 600 lines per minute are obtainable. The speed is variable depending on the number of carbons desired. The printer can print any of 51 characters. The checking features include an odd-even and 120-character check on the reading from the tape, a check on the internal memory, and a check on the proper firing of the print hammers.

The Uniprinter is an electric typewriter, identical in its action to the Supervisory Control Printer except its input as magnetic tape, recorded at the low density of 20 characters/inch. The Uniprinter is used where low volume printing of extreme flexibility in form is desired. The printing rate is about 8-10 characters/second and the typing format is controlled by typewriter control symbols recorded directly on the tape with the data. In this manner, the typing format can be as variable as that obtainable from a typist.

Tape To Punched Card:

The magnetic Tape-To-Card Converter produces an 80 column punched card from each 10-word item recorded on tape. A plugboard allows the selection of any 80 digits of the item to be entered in any of the 80 card columns. The checking features are the odd-even check and a 120-check on reading from tape, memory check, and a comparison of the punched card with the data stored in the memory. The conversion takes place at a rate of 120 cards/minute with tape recorded at a density of 128 characters/inch.

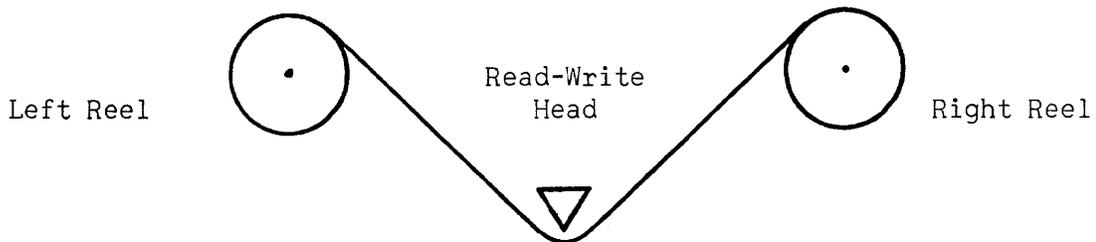
Tape To Computer:

This is accomplished by Uniservos which are described in Section IV.

SECTION IV

Uniservo

Communication between the Univac and the auxiliaries described in Section II and III is established through magnetic tape. The Uniservo is the means by which the Univac reads or records magnetic tape. As many as ten Uniservos may be connected to the Univac at the present time. They are numbered from 1 to 9, the tenth being labeled the -. A Uniservo may read or write tape at the discretion of the program, and is two-directional in its action. In defining the direction of tape motion we shall use the simplified picture shown below.



A reel of tape is always mounted on the left reel. It is connected to a pre-threaded leader which is permanently fastened to the right reel. Removal of a reel of tape and mounting a new reel takes about 1/2 minute. When tape is moving from the left reel to the right reel, we say the tape is moving in the forward direction. When tape moves from the right reel to the left, the tape is said to be moving backwards.

SECTION V

Instruction List E

5n m, This instruction causes the computer to write on Uniservo n ($n = 1, 2, \dots, 9, -$) the block, 60 words, from memory cells $m, m - 1, \dots, m - 59$. m must be a multiple of ten. The writing is at the high tape density of 128 characters/inch, with the tape moving in a forward direction. The write instruction is carried out in three steps:

- 1) The computer determines whether:

- a. Another write instruction is not still in progress
- b. The last write was properly executed
- c. Uniservo n is free (not engaged in a rewind or read instruction)

If all of the above queries are answered affirmatively, it proceeds to step two; otherwise, it remains at step one until all queries are answered yes. These queries are called the write interlock tests.

- 2) The block in m through m + 59 is transferred to the 60-word output register, r0, and Uniservo n is started. At this point the central computer is released to continue executing instructions, the output control circuits taking over control.
- 3) The block in r0 is recorded on tape, first the word that came from cell m, then m + 1, etc., and last to be recorded is the word from cell m + 59. Within each word, the first character recorded is from digit position 1, then, 2, and finally position 12. After the last word is recorded the tape stops and Uniservo n is free as also are the output circuits.

7n m, This instruction is identical to the 5n m instruction, except the write density is 20 characters to the inch. This density is used only when the tape is to be printed on the Uniprinter.

ln 000, This instruction causes the computer to read into rI in a forward direction (the block just to the left of the read-write head on Uniservo n). The read instruction is carried out in two steps.

- 1) The computer determines whether:
 - a. Another read instruction is not still in progress
 - b. The last read was properly executed
 - c. Uniservo n is free (not engaged in a rewind or write instruction)

If all of the above queries are answered affirmatively, the computer proceeds to step two, otherwise it remains at step one until all queries are answered yes. These queries are called the read interlock tests.

- 2) Uniservo n started, the tape moves in a forward direction, the central computer is then released so that it may continue executing instruction, and the input control circuits take over control of the read order. As each character passes under the read-write head, it is read and this information stored in the block lengths input register, rI. At the conclusion of the read instruction rI will appear exactly as r0 appeared when the block was written. At this point the tape stops and Uniservo n and the input circuits are freed.

2n 000, This is the backward read instruction, and is identical to the 1n 000 instruction, except that the block read into rI is the one to the right of the read-write head, the tape moving backwards. Even though the block is read backwards, its appearance in rI is the same as though it were read forward.

30 m, This instruction causes the computer to transfer the block in rI to memory cells m, m + 1, ..., m + 59. rI is cleared only after the transfer. m must be a multiple of ten.

This instruction is carried out in two steps:

- 1) The read interlock tests (described under the 1n 000 instruction) are performed. If they are answered yes, step 2 is executed.
- 2) The 60 words stored in rI are transferred to the memory beginning with cell m, the last word in rI going to cell m + 59. After completion of the transfer, the computer is released to continue executing instructions.

3n m, This is a composite instruction, performing first a 30 m, and then a 1n 000.

4n m, This is a composite instruction, performing first a 30 m, and then a 2n 000.

6n 000, This is the rewind instruction, causing all of the tape on the right hand reel to be passed over to the left reel. This instruction is executed in two steps:

- 1) The write interlock tests are performed.
- 2) The Uniservo n is started, moving tape backwards. At this point the central computer is released and the rewind control circuits continue the rewind process.

At the conclusion of the rewind, the tape may be read forward or written upon.

8n 000, This is the Rewind with interlock instruction. Its action is identical with 6n 000 instruction, except, at the completion of the rewind an interlock is set for Uniservo n.

Whenever Uniservo n is called for again to either read, write or rewind it always sends back a "busy" signal, stopping the computer on the appropriate interlock tests. This instruction is used when a rewound tape is to be removed and a new tape mounted in its place. The removal of the reel of tape and the mounting of a new reel removes the busy signal for Uniservo n.

The use of buffer storages between the computer and the tape units and the separate read and write control circuits permits high speed input and output operations, as the computer is held up only for the small amount of time necessary to perform the interlock tests and fill the output buffer or empty the input buffer. Further, the interlocks prevent one from trying to use information which is being written or read until the write or read is completed.

The following problem will illustrate the use of these orders.

A tape on Uniservo 2 contains a series of two-word inventory records.

| |
|--------------|
| Stock Number |
| Inventory |

The number of such items is unknown but a Z sentinel follows the last item. An equal number of two-word items

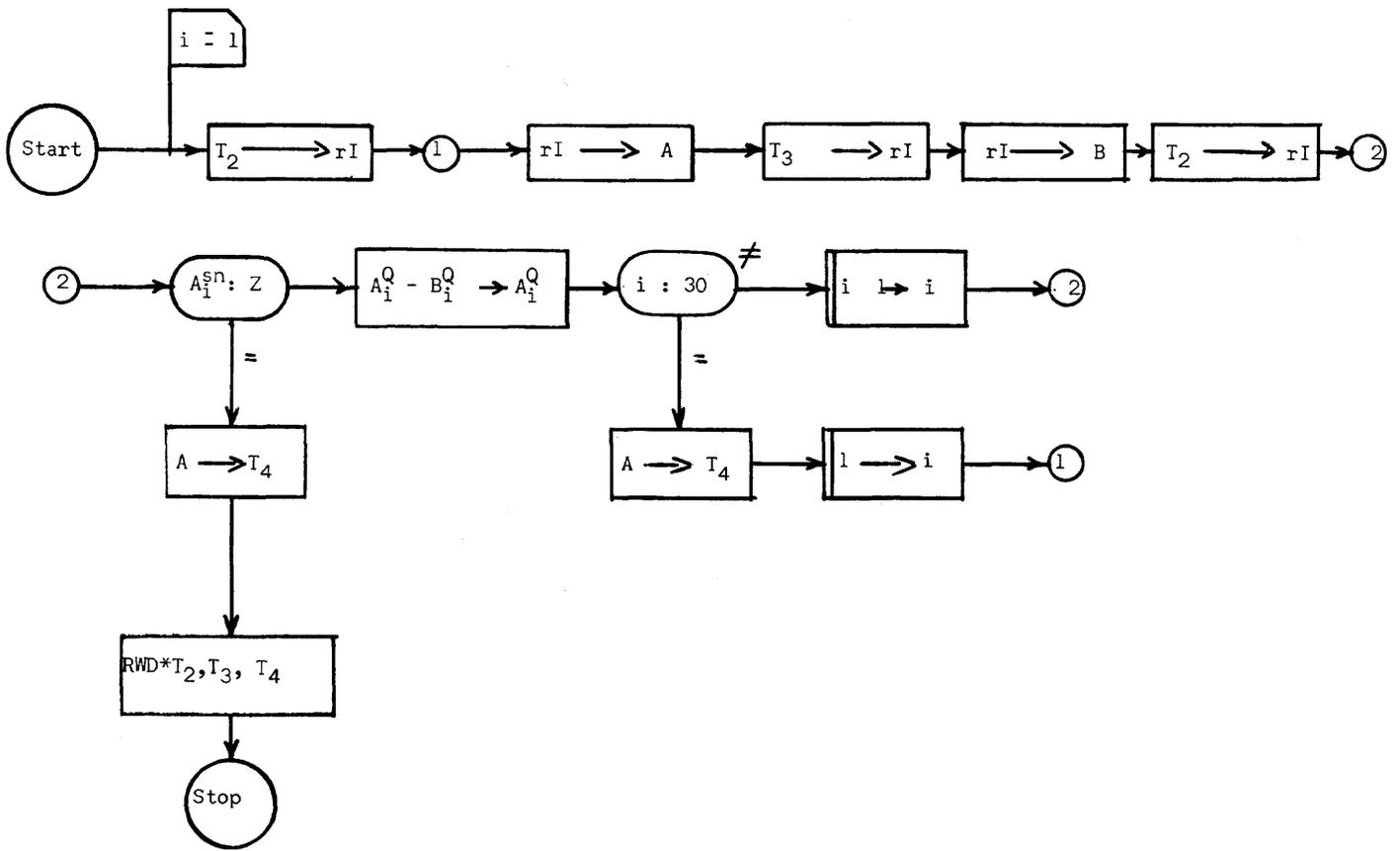
| |
|---------------|
| Stock Number |
| Quantity Used |

are recorded on Tape 3. The stock numbers are assumed to be in the same sequence on both tapes. Produce a corrected inventory on Tape 4.

Let us label Tape 2 as T_2 , a block (30 items) from this tape, A, each of the thirty items A_i , and the stock number of an item A_i^{sn} , the quantity A_i^Q

Tape 3 is labeled T_3 , a block from the tape, B, an item of the block B_i and the stock number and quantity B_i^{sn} , B_i^Q .

| | | | | | |
|-----|----|-----|----|-----|--|
| 000 | 12 | 000 | | | $T_2 \longrightarrow rI$ |
| 001 | 33 | 100 | 00 | 000 | $rI \longrightarrow A, T_3 \longrightarrow rI$ |
| 002 | B | 100 | 32 | 200 | $rI \longrightarrow B, T_2 \longrightarrow rI$ |
| | | | L | 018 | |
| 003 | 00 | 000 | Q | 015 | Transfer control if $A_i^{sn} = Z$ |
| 004 | B | 101 | S | 201 | } $A_i^Q - B_i^Q \longrightarrow A_i^Q$ |
| 005 | H | 101 | B | 002 | |
| 006 | L | 019 | Q | 012 | Transfer control if $i = 30$ |
| 007 | A | 020 | H | 002 | |
| 008 | B | 004 | A | 021 | } $i + 1 \longrightarrow i$ |
| 009 | H | 004 | B | 005 | |
| 010 | A | 020 | H | 005 | |
| 011 | 00 | 000 | U | 002 | |
| 012 | 54 | 100 | B | 024 | $A \longrightarrow T_4$ |



| | | | | | |
|-----|-----|-----|-----|-----|---|
| 013 | H | 002 | V | 022 | 1 → i |
| 014 | W | 004 | U | 001 | |
| 015 | 54 | 100 | 82 | 000 | A → T ₄ |
| 016 | 83 | 000 | 84 | 000 | Rewind W/Interlock T ₂ , T ₃ , T ₄ |
| 017 | 90 | 000 | 00 | 000 | Stop |
| 018 | ZZZ | ZZZ | ZZZ | ZZZ | |
| 019 | B | 158 | L | 018 | |
| 020 | 000 | 002 | 000 | 000 | |
| 021 | 000 | 002 | 000 | 002 | |
| 022 | B | 101 | S | 201 | |
| 023 | H | 101 | B | 002 | |
| 024 | B | 100 | L | 018 | |

Student Exercise:

A tape on Uniservo 1 contains a series of two word meter consumption items

| |
|----------------|
| Account Number |
| Consumption |

The number of such items is unknown but a Z sentinel follows the last item. Produce and print the following table:

| Consumption Range | Consumption | Accounts |
|-------------------|-------------|----------|
| 1-100 | 3600 | 500 |
| 101-500 | - | - |
| 501-1000 | - | - |
| 1001-over | - | - |

SECTION VI

Filling Register I

In most phases of complex commercial applications of UNIVAC, data from several different sources must be brought together before computation on this data can proceed. The information coming from these different sources are recorded on magnetic tape and each separate type of data is assigned a Uniservo which serves as the transport device, enabling the information on the tape to be brought into the central computer in units of sixty words, one block. As noted in the instruction code pertaining to the input orders, Section V, all information from the tapes must pass into the sixty-word Register I, and, thus, only a block of data from one Uniservo may be read into the computer at a given time. There is an instruction, the $3n\ m$ (or $4n\ m$, for backward reading) whereby the programmer may transfer the block of data present in rI into the memory for processing, and simultaneously order any particular Uniservo to read another block of data into rI, this reading being done independently of the operation of the central computer which is free to begin calculations on the block just transferred from rI.

Thus, if the computation necessary to process a block of information is long enough, the time to read the information into the computer may be completely absorbed by it. Or, if the amount of time taken up in computing on the data is less than the time required to read the data into the machine, the computing time is completely absorbed by the tape reading time. This is true, of course, only if continuous read instructions, $3n\ m$ or $4n\ m$, are given. For the combination $ln\ 000$ followed by $30\ m$, the lapse of time between the execution of the left instruction and the right instruction will be of the order of 100 milliseconds, the time required to fill rI. From the standpoint of elapsed computer time, it is desirable to do continuous read instructions.

Where the processing to be done consists of bringing information into the computer from several different tapes and in an order which is not known in advance to the programmer, that is, in essentially a random fashion (unlike the example of Section V where the order of reading is a block from tape 3 for each block from tape 2), how is the programmer to make sure that the data will be brought in at the right time and from the right tape?

For example, consider the problem basic to almost all commercial applications:

Information, consisting of a series of items containing a serial number, are recorded on tape in ascending order by this serial number. Two such sets of items (hereafter called A and B) are to be merged or interfiled so as to produce one tape containing all the items present on both tapes, but arranged in ascending order. This is the problem presented in Chapter 5 Section IV, adapted for tape input and output.

The first block from each tape is brought into the computer, and their first

items compared. The item with the smallest serial number is then transferred to the first position of an output block. If, for example, the lowest item in the comparison was from the A set of data, this item is placed in the output block. The next A item of the block is compared with this first B item and the smallest of these two items is sent to the second position of the output block. When the output block is filled, that is, when sufficient items have been transferred so that the output block contains 60 words, it is written on the output tape. The next lowest item transferred will be into the first output item position again. Soon, one of the input blocks will be exhausted, all of its items having been transferred to the output block. Therefore, we must bring in a new block of these items from tape. If we do the transfer from tape storage to computer by the sequence ln 000 -30 m, the computer must wait for approximately 100 milliseconds before it can execute the transfer from rI to the memory. But, if rI already contained the right block of data, we would be able to continue the processing by waiting only for the 5.5 millisecond transfer from rI to the memory. Since the order in which information from the two tapes is to be read is determined by the data on the tapes, it would seem that the programmer cannot do continuous read instructions for this kind of problem. Actually, there are several methods by which rI can be kept filled with the proper block at the proper time, one of which, the simplest, will be described.

This method, called "Preselection", will be illustrated by doing the two-tape merge problem:

A tape on Uniservo 2 contains a series of ten-word "A" items, arranged on tape in ascending order by a serial number which is the first word of each item. The number of such items is unknown, but a Z sentinel follows the last item.

A similarly arranged set of "B" items are recorded on a tape mounted on Uniservo 3. Interfile the items on the tapes, writing the merged data on tape 4.

Consider the possible appearance of the first six serial numbers of each tape (one Block):

| <u>Tape 2</u> <u>Serial Numbers</u> | <u>Tape 3</u> <u>Serial Numbers</u> |
|--|--|
| 1025 | 96 |
| 1027 | 876 |
| 1106 | 1541 |
| 2257 | 1995 |
| 2450 | 2630 |
| 2451 | 3001 |

It is apparent, that in this merging process, the 6th A-item must appear on the output tape before the sixth B item, as it has a lower item serial number. But this means, since the items on each input tape are in ascending order by their serial numbers, that the A block will be used up first. Thus, rI should be filled with the next block from tape 2. This information can be determined before any processing is done on the A and B items, thus allowing one to give the instructions to fill rI as soon as the blocks are in the computer.

The flow chart and coding for this problem are shown below. The notation used is:

T_2 Specifies the tape on Uniservo 2.

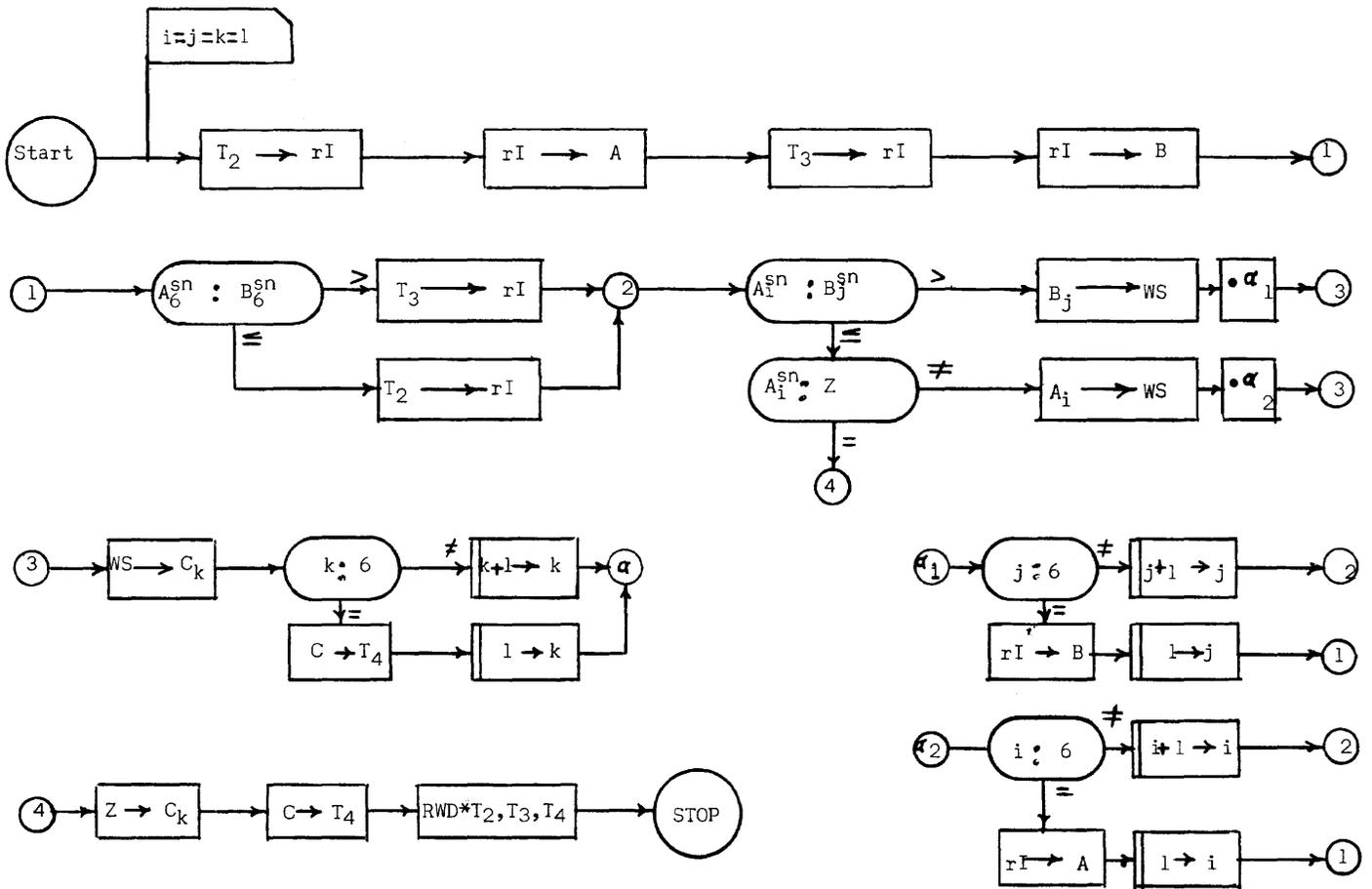
$T_2 \rightarrow rI$ Indicates that the next block on tape 2 is read (forward direction) into register I

$rI \rightarrow A$ The block in rI becomes the block A.

A_i The i th item of the block A, $i = 1, 2, \dots, 6$.

A_i^{sn} The serial number of the i th A item.

Similar notation for $T_3, B, B_j, B_j^{sn}, C_k, C, T_4$, with $C \rightarrow T_4$ meaning the block C is written on tape 4.



| | | | | |
|-----|----|-----|----|-----|
| 000 | 12 | 000 | | |
| 001 | 30 | 200 | 33 | 100 |
| 002 | B | 150 | 00 | 000 |
| 003 | 00 | 000 | L | 250 |
| 004 | 12 | 000 | T | 005 |
| 005 | 13 | 000 | U | 006 |
| 006 | B | 100 | 00 | 000 |
| 007 | 00 | 000 | L | 200 |
| 008 | L | 040 | T | 024 |
| 009 | Y | 100 | Q | 033 |
| 010 | H | 014 | B | 041 |
| 011 | Z | 300 | 00 | 000 |
| 012 | L | 039 | B | 011 |
| 013 | A | 041 | Q | 022 |
| 014 | - | | H | 011 |
| 015 | B | 018 | | |
| 016 | 00 | 000 | L | 042 |
| 017 | A | 041 | Q | 020 |
| 018 | Y | 100 | H | 018 |
| 019 | 00 | 000 | Z | 100 |
| 020 | 30 | 100 | U | 006 |
| 021 | H | 018 | B | 043 |
| 022 | 54 | 300 | U | 002 |
| 023 | H | 011 | B | 044 |
| 024 | Y | 200 | U | 014 |
| 025 | H | 014 | B | 045 |
| 026 | B | 029 | U | 011 |
| | | | L | 046 |

$T_2 \rightarrow rI$
 $rI \rightarrow A, T_3 \rightarrow rI$
 $rI \rightarrow B$

Transfer control if $A_6^{sn} > B_6^{sn}$
 $T_2 \rightarrow rI$

$T_3 \rightarrow rI$

Transfer control if $A_i^{sn} > B_j^{sn}$

Transfer control if $A_i^{sn} = Z$
 $A_i \rightarrow WS$ (register Y)

α_2

$WS \rightarrow C_k$

Transfer control if $k = 6$
 $k + 1 \rightarrow k$

Variable connector α

Transfer control if $i = 6$

$i + 1 \rightarrow i$

$rI \rightarrow A$
 $l \rightarrow i$

$C \rightarrow T_4$
 $l \rightarrow k$

$B_j \rightarrow WS$ (register Y)
 α_1

| | | | | |
|-----|-----|-----|-----|-----|
| 027 | 00 | 000 | | |
| 028 | A | 041 | Q | 031 |
| 029 | Y | 200 | H | 029 |
| 030 | 00 | 000 | Z | 200 |
| 031 | 30 | 200 | U | 006 |
| 032 | H | 029 | B | 047 |
| 033 | B | 011 | U | 002 |
| 034 | Y | 040 | H | 035 |
| 035 | - | | 00 | 000 |
| 036 | 54 | 300 | | |
| 037 | 83 | 000 | 82 | 000 |
| 038 | 90 | 000 | 84 | 000 |
| 039 | Z | 350 | 000 | 000 |
| 040 | ZZZ | ZZZ | B | 011 |
| 041 | 000 | 010 | ZZZ | ZZZ |
| 042 | Y | 150 | 000 | 000 |
| 043 | Y | 100 | Z | 100 |
| 044 | Z | 300 | Z | 100 |
| 045 | 00 | 000 | B | 011 |
| 046 | Y | 250 | U | 026 |
| 047 | Y | 200 | Z | 200 |
| | | | Z | 200 |

Transfer control if $j = 6$

} $j + 1 \longrightarrow j$

$rI \longrightarrow B$
 } $1 \longrightarrow j$

$Z \longrightarrow Ck$

} Rewind with interlock T_2, T_3, T_4
 Stop

SECTION I

Introduction

The purpose of Chapters 8, 9, and 10 is to provide the reader with a basic understanding of the logical construction of general purpose digital computers. This knowledge is important when questions concerning the evaluation of different computers arise, and will also aid the reader in broadening his understanding of the entire art.

SECTION II

Binary Representation

We have been accustomed since our early grade school days to think about numbers in the Arabic notation. In this notation, because of our years of usage, 1076, is immediately significant to us as one thousand and seventy-six. And yet, the Arabic numerals are not the only way of writing numbers. As an example, the reader is not frequently exposed to the number MDCCCCLIIII = 1954 commonly used as date marks in cornerstones or publications.

Implicitly understood in our writing the number 1076, is that it is a shortcut scheme for saying $1 \times 1000 + 0 \times 100 + 7 \times 10 + 6 \times 1$. Thus, the Arabic notation is said to be a decimal or base 10 number system because one of the factors in each product is a multiple, or power of ten:

$$\begin{array}{rcl}
 1000 & 10 \times 10 \times 10 & = 10^3 \\
 100 & 10 \times 10 & = 10^2 \\
 10 & 10 & = 10^1 \\
 1 & \text{(Defined as)} & = 10^0
 \end{array}$$

In writing 1076, then, we save ourselves effort by dropping the various multiples of ten and simply write their coefficients 1, 0, 7, 6, where the column in which the coefficient appears indicates the appropriate ten's multiple. As many as ten different coefficients are possible for any column: i.e. 0, 1, 2, 3, 4, 5, 6, 7, 8, or 9.

As mentioned in Chapter 1, and illustrated by the chapters following it, a number is the basic element processed by a computer. If these numbers are expressed by Arabic numerals, we must have devices which can represent any one of the ten digits 0, 1, ...9; that is, these devices must have ten stable states easily and accurately distinguished from one another. This is an easy accomplishment if our computer is mechanical in nature. One of the simplest of such devices is the notched wheel found in speedometers (more accurately, the odometer part) or adding machines. The angular position of the wheel as referenced to a fixed mark represents one of the ten numerals. In fact, the Harvard (University) Mark 1 computer, completed in 1944 and the first successful general purpose digital computer, is probably the most complex application of the notched wheel. Unfortunately, mechanical number representation implies slow action and great bulk. Electronic components give much faster operation and the Eniac computer built by Eckert, Mauchly and others at the University of Pennsylvania in 1946 was the first

electronic digital computer. It represented numbers decimally, but because electronic gear (at the present state of the art) does not lend itself naturally to decimal notation nearly 18,000 vacuum tubes were necessary to achieve a computer of only 20-word storage.

Electronic elements lend themselves most naturally to a two-stable state scheme—a vacuum tube either conducts current or it does not, a wire is at one voltage level or it is not. Is it possible to represent numbers, then, using only two numerals? It is, and this notation is called binary representation. Consider the successive powers of two:

$$\begin{aligned}
 2^0 & \text{ (Defined as) } = 1 \\
 2^1 & = 2 = 2 \\
 2^2 & = 2 \times 2 = 4 \\
 2^3 & = 2 \times 2 \times 2 = 8 \\
 2^4 & = 2 \times 2 \times 2 \times 2 = 16
 \end{aligned}$$

etc.

It is possible to express any number as a sum of these powers of two, for example:

$$\begin{aligned}
 7 & = 4 + 2 + 1 & 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
 13 & = 8 + 4 + 1 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0
 \end{aligned}$$

As in the decimal notation, we can save ourselves effort by simply writing down the coefficients (which are either 1 or 0) and not the appropriate power of two by which it is multiplied. Thus, in binary notation:

| <u>Decimal Number</u> | <u>Binary Equivalent</u> |
|-----------------------|--------------------------|
| 7 | 111 |
| 13 | 1101 |
| 116 | 1110100 |

SECTION 111

Binary Arithmetic

The rules of binary addition are quite simple,

| | | | |
|--|--|--|---|
| $ \begin{array}{r} 0 \\ + 0 \\ \hline 0 \end{array} $ | $ \begin{array}{r} 0 \\ + 1 \\ \hline 1 \end{array} $ | $ \begin{array}{r} 1 \\ + 0 \\ \hline 1 \end{array} $ | $ \begin{array}{r} 1 \\ + 1 \\ \hline 0 \text{ and } 1 \text{ carry} \end{array} $ |
|--|--|--|---|

The following examples will make evident the ordinary arithmetic behind these rules:

$$\begin{aligned}
 11 & = & 1 \times 8 & + & 0 \times 4 & + & 1 \times 2 & + & 1 \times 1 \\
 + 4 & = & 0 \times 8 & + & 1 \times 4 & + & 0 \times 2 & + & 0 \times 1 \\
 15 & = & (1 + 0) \times 8 & + & (0 + 1) \times 4 & + & (1 + 0) \times 2 & + & (1 + 0) \times 1 \\
 & = & 1 \times 8 & + & 1 \times 4 & + & 1 \times 2 & + & 1 \times 1
 \end{aligned}$$

$$\begin{aligned}
13 &= 1 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 \\
+ 7 &= 0 \times 8 + 1 \times 4 + 1 \times 2 + 1 \times 1 \\
\hline
20 &= (1 + 0) \times 8 + (1 + 1) \times 4 + (0 + 1) \times 2 + (1 + 1) \times 1 \\
&= 1 \times 8 + 2 \times 4 + 1 \times 2 + 2 \times 1 \\
&= 2 \times 8 + 0 \times 4 + 2 \times 2 + 0 \times 1 \\
&= 1 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 0 \times 1
\end{aligned}$$

Remembering the rules for binary addition, these examples are simply:

$$\begin{array}{r}
11 = 1011 \\
+ 4 = 0100 \\
\hline
15 = 1111
\end{array}
\qquad
\begin{array}{r}
13 = 1101 \\
+ 7 = 0111 \\
\hline
20 = 10100
\end{array}$$

Occasionally, we use the concept of complements in our ordinary decimal arithmetic. For example, the old rule of casting out nines to check addition. The binary complement of a number is obtained by subtracting that number from the next highest power of two. The binary complement of 7 is 1, since $8 - 7 = 1$. The binary complement of a number expressed in binary notation is easy to obtain by applying the following rule:

Replace all ones by zeros and all the zeros by ones; then add 1 in the least significant column.

Thus, $6 = 110$, replacing ones by zeros and zeros by ones, we have 001 and adding one, $010 = 2 = 8 - 6$, the binary complement of 6.

Complements are used in the subtraction of binary numbers because of their simplicity over direct subtraction. To illustrate subtraction using complements consider the following case:

$$7 - 3 = 7 + (8 - 3) - 8 = 4$$

Two's complement of 3

Since we can obtain the two's complement in a simple manner, we can do subtractions by performing additions if we can subtract the power of two used in the complement from the sum obtained. This also is easily accomplished by dropping the carry from the left-most column.

$$\begin{array}{r}
7 = 111 \\
- 3 = 101 \quad (\text{Two's complement of 3}) \\
\hline
4 = 100 \quad (\text{Dropping carry from left})
\end{array}$$

$$\begin{array}{r}
 15 \\
 - 6 \\
 \hline
 9
 \end{array}
 \qquad
 \begin{array}{r}
 1111 \\
 1010 \\
 \hline
 1001
 \end{array}
 \begin{array}{l}
 \text{(Two's complement of 6 is 0110)} \\
 \text{(Dropping carry from left)}
 \end{array}$$

Multiplication of numbers expressed in binary notation is extremely simple. The multiplication table is:

$$\begin{array}{r}
 0 \\
 0 \\
 \hline
 0
 \end{array}
 \qquad
 \begin{array}{r}
 1 \\
 0 \\
 \hline
 0
 \end{array}
 \qquad
 \begin{array}{r}
 0 \\
 1 \\
 \hline
 0
 \end{array}
 \qquad
 \begin{array}{r}
 1 \\
 1 \\
 \hline
 1
 \end{array}$$

For example:

$$\begin{array}{r}
 7 \\
 3 \\
 \hline
 21
 \end{array}
 =
 \begin{array}{r}
 111 \\
 011 \\
 \hline
 111 \\
 111 \\
 000 \\
 \hline
 10101
 \end{array}
 = 21$$

Note, that at every step we are performing addition only. A means for writing each of the partial products one column to the left at each step is the only extra feature required for the adder to perform multiplication.

The simplest way of performing division is to use the long division method taught in the lower grades. In this method, the divisor is subtracted repeatedly from the dividend, a one being added to the quotient (which is set initially to zero) for each subtraction. The following example will illustrate the method:

| In Decimal Notation | In Binary Notation |
|---|---|
| $6 \overline{)18}$ | $00110 \overline{)10010}$ |
| $6 \overline{)18}$ $\underline{6}$ 12 | $00110 \overline{)10010}$ $\underline{00001}$ 00110 $\underline{01100}$ |
| $6 \overline{)18}$ $\underline{6}$ 12 $\underline{6}$ | $00110 \overline{)10010}$ $\underline{00010}$ 00110 $\underline{01100}$ 00110 |
| $6 \overline{)18}$ $\underline{6}$ 12 $\underline{6}$ 6 $\underline{6}$ 0 | $00110 \overline{)10010}$ $\underline{00011}$ 00110 $\underline{01100}$ 00110 $\underline{00110}$ 00110 $\underline{00000}$ |

In summary then, all the arithmetic operations, addition, subtraction, multiplication, and division, can be performed on numbers expressed in binary notation provided we can construct an adder, complemeter, and shift mechanism. In Chapter 10 the construction of these devices will be discussed.

SECTION IV

Coded Decimal Notation

The binary method of representing numbers was described in Section 11. This is the almost universal representation used in large computers. This binary notation may be used in two forms. In the first, called "pure binary," the ones and zeros making up a number have a place value that represents some power of two. For example, in the binary number 10110, the left-most one has the value 2^4 or 16. The other method is called the "coded decimal" notation. Any number between 0 and 9 may be represented by a minimum of four binary columns as the following table shows.

| <u>Decimal Digit</u> | <u>Binary Equivalent</u> |
|----------------------|--------------------------|
| 0 | 0000 |
| 1 | 0001 |
| 2 | 0010 |
| 3 | 0011 |
| 4 | 0100 |
| 5 | 0101 |
| 6 | 0110 |
| 7 | 0111 |
| 8 | 1000 |
| 9 | 1001 |

Now consider the decimal number 147. If this number were expressed in binary notation, it would appear as 10010011. However, we could represent each of the individual decimal digits in binary fashion and yet retain their decimal value as regards position. Thus:

| | | |
|------|------|------|
| 0001 | 0100 | 0111 |
| 1 | 4 | 7 |

This coded decimal notation is more convenient than pure binary to the user of a computer, primarily because we can read the numbers easily. With very little practice one can recognize the coded decimal number 0010 1000 0111 0110 as being 2876, but the same amount of practice does not yield equal results for its pure binary form 101100111100.

Of course, we do not normally "look" at numbers in either binary or coded decimal form. But when the input data is to be prepared for a computer, we would prefer not to have to convert decimal numbers to binary form acceptable to the computer, nor should we like to read the output of our computer expressed binary-wise.

For computers operating in coded decimal form, information may be introduced to or removed from the computer in decimal form directly. With pure binary computers, input and output data must be converted through a program to decimal form. Because of this time consuming conversion data processing computers use coded decimal notation.

Many modifications of the coded decimal notation described above are in common use. These modifications are designed to impart certain desirable arithmetic properties to the coded numbers. Several of these will be described.

In Univac, the excess-3 coded decimal is used. Here, a decimal digit is represented in the binary scheme by a number larger by 3. Thus, 6 would be represented in coded decimal form by $6 + 3 = 9 = 1001$. This modification has two desirable properties: Ease in obtaining the nine's complement (replace one's by zeros, zeros by one's-nine's complements of $6 = 9 - 6 = 3 = 0110$); and the sum of two XS-3 numbers produce a binary carry when their decimal sum produces carry.

The so-called 2*421 scheme also has these same two features of XS-3. In this scheme the numbers below 5 are in normal binary notation, but 5 and above are coded as having a 2* plus the appropriate 421 code.

The bi-quinary method uses seven binary columns per decimal digit. Its feature is that only two binary one's are present for any digit.

The following table will illustrate these representations:

| Decimal | Coded Decimal Equivalent | | | |
|---------|--------------------------|------|--------|------------|
| | Straight | XS-3 | 2* 421 | Bi-quinary |
| 0 | 0000 | 0011 | 0000 | 01 00001 |
| 1 | 0001 | 0100 | 0001 | 01 00010 |
| 2 | 0010 | 0101 | 0010 | 01 00100 |
| 3 | 0011 | 0110 | 0011 | 01 01000 |
| 4 | 0100 | 0111 | 0100 | 01 10000 |
| 5 | 0101 | 1000 | 1011 | 10 00001 |
| 6 | 0110 | 1001 | 1100 | 10 00010 |
| 7 | 0111 | 1010 | 1101 | 10 00100 |
| 8 | 1000 | 1011 | 1110 | 10 01000 |
| 9 | 1001 | 1100 | 1111 | 10 10000 |

SECTION V

Alphabetic Representation and Checking Codes

Although numbers are the basic processing element in computers, the earlier chapters have made evident the desirability of permitting alphabetic characters and punctuation symbols to be intelligible to the computer. This may be accomplished in several ways with computers using a coded decimal notation.

For a full alpha-numerical representation, a character (= digit) can have any one of 36 possible values: The numbers 0 through 9 or the letters A through Z. By adding the minimum of two additional binary columns to our four bit coded decimal number we can represent 64 different symbols, giving an extra 28 codes for punctuation and other symbols. In the Univac, for example, the letter A is represented by the XS-3 code for one plus a 01 "zone" code. Other examples are shown in the table.

| Character | Binary Representation |
|-----------|-----------------------|
| 3 | 000110 |
| 9 | 001100 |
| D | 010111 |
| P | 101010 |
| T | 110110 |

Some computers represent alpha-numeric quantities by means of paired digits or paired words, special commands being provided for operations with the alpha-numeric "words."

In either scheme, two decimal digits are used to represent the full alpha-numeric gamut:

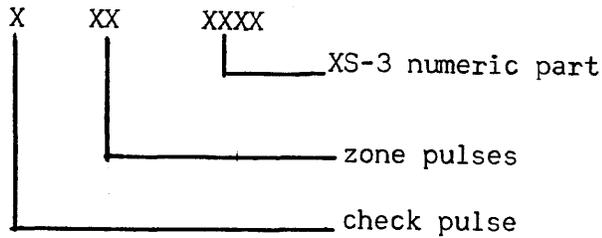
| Character | Coded Representation |
|-----------|----------------------|
| 0 | 00 |
| 3 | 03 |
| 9 | 09 |
| B | 11 |
| E | 14 |
| R | 26 |

In one such computer the digits within a word are paired off, a symbol in the sign position being sufficient to tell the computer the number being acted upon is a coded alpha-numeric quantity. Another uses two words whose addresses are related. The same numbered columns in the two words being associated to form the alpha-numeric character.

The tendency in new computers is to provide automatic safeguards against an error, produced by equipment failure, to propagate itself undetected. Indeed, the principles of digital computation demands complete accuracy if the results of a calculation run are to be meaningful. This is a point often neglected by prospective computer users: They fail to realize that an error may occur in the control portion of the device as well as the arithmetic portion.

One common checking means is to produce a certain amount of redundancy in the coded number representation. One example already described is the bi-quinary coded decimal notation. Here, a correct character requires a single binary one in the "Bi" part and a single one in the "Quinary" part. Loss or gain of a one in either part represents an illegal character which can be detected by the computer.

In the Univac, an extra binary column is attached to each character. This column is called the check pulse and is assigned in such a manner that an odd number of binary one's is present for each valid character. As the characters are processed, their binary one's are counted. If any even count results, it means a one has been picked up or lost and the computer registers an error and stops. Thus, the full Univac coded decimal representation of a character is comprised of seven binary columns.



As an example the zero has this binary code

1000011

The P

0101010

SECTION 1

Introduction

In Chapter 7 a way was developed whereby the numbers and alphabetic characters involved in data processing problems could be represented by a simple pattern of ones and zeros (the binary code). Further, we saw that all of our usual arithmetic manipulations could be performed on numbers expressed in this binary code. As was pointed out in that chapter, the binary notation is in common use because electronic gear is easily adapted to this two-valued method of representation. In this chapter we shall briefly describe the various techniques used in storing or "memorizing" such binary information in computers. Storage of binary information requires bi-stable devices. By this, we mean devices exhibiting two, easily-distinguishable states and which when placed in any one of the states will remain in that state as long as desired.

Before describing the storage elements now in use, it is necessary to recognize that we can store information in two basic ways, either in static or dynamic mode.

In static storage the information to be stored is distributed throughout space. While in dynamic storage, this information is distributed in time. Perfect, and yet homely, examples of each type of storage are:

For static storage, the printed page. Here, all of the data stored (in the form of words) is available to us at the same instant of time. We need only direct our eye to the desired spot on the page to select any portion of the information.

For dynamic storage, we need go no further than spoken conversation. Here, we can obtain all of the information (in the form of sounds) by waiting in time for our informant to finish speaking.

Computers are often described as being serial or parallel in operation. By serial operation it is meant that the binary digits comprising the computer word are extracted from the memory, manipulated, and placed back into the memory sequentially in time. Thus, in serial mode all digits pass through essentially one set of circuits. This is economic of circuitry. While in parallel operation, all binary bits of a word are handled simultaneously in time. This requires considerable increase in circuitry, but other things being equal, it is the fastest in operation. These two modes of operation are in the main a result of the type of memory used. Static storage lends itself naturally to parallel operation and dynamic storage to serial operation. It should be noted, however, that there are some exceptions to this statement.

SECTION 11

Static Storage Devices

Relay

One of the earliest, and probably simplest, electronic devices used in computer memories is the relay. As shown in Figure 1, this device consists of a steel arm sprung to avoid electrical contact with a read-out wire. The arm may be brought into contact with the wire by applying a current through the electromagnet which draws the arm downwards. A latch mechanism can hold the arm down indefinitely if desired, and then the relay does not require continuous current flow through the magnet coils. The two stable states of the relay are contact, placing a current on the read-out wire (binary 1); or no contact, placing no current on the wire (binary 0). The relay can be made to change state reliably in from 1 to 10 milliseconds. Relays are not in common use now as memory elements since they are relatively slow, bulky, and quite expensive. Several early computers using them as memory elements are still in operation.

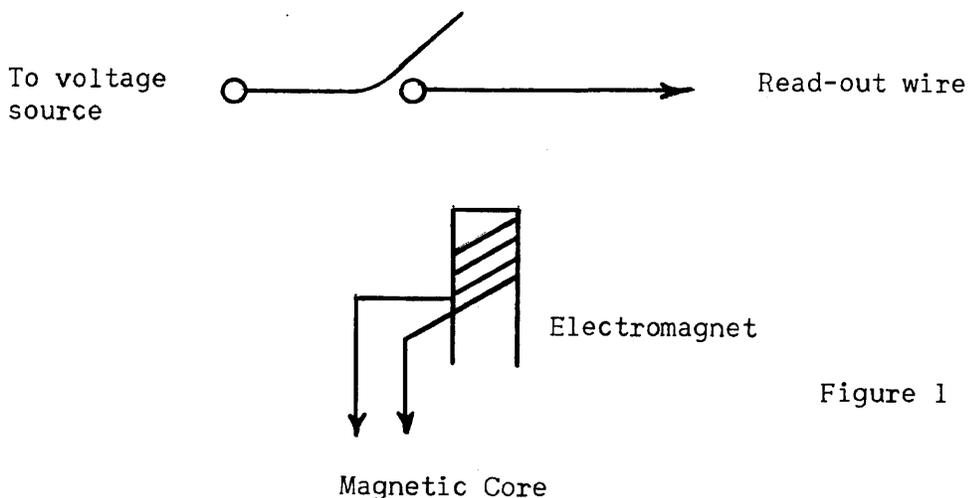
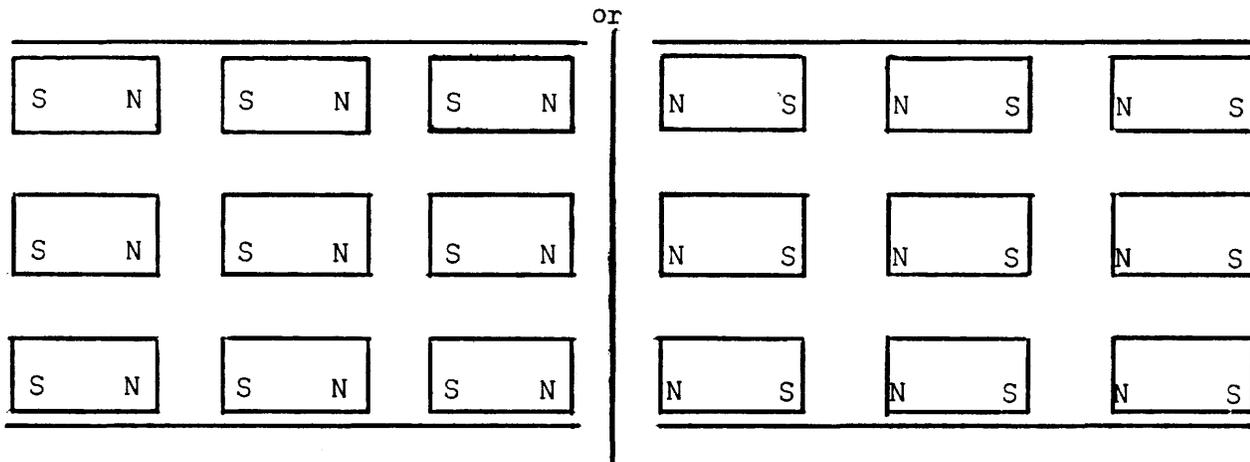


Figure 1

The core may be thought of as being composed of elementary bar magnets oriented in one of two directions parallel to its axis.



In practise the core is constructed in the form of a torus with three coils wound about it: The input winding, output winding and a probe winding.

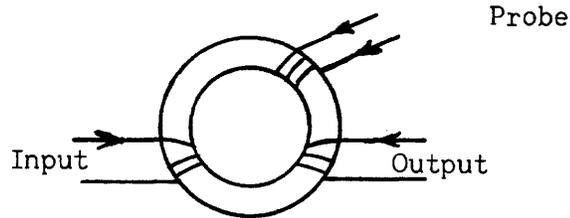


Figure 2

The core has two stable states and these may be considered as representing binary one and binary zero. To read into a core, a pulse is applied to the input coil. The direction of the current will determine which state the coil will assume. When the current is removed

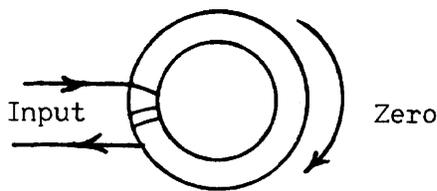


Figure 3A

Magnetic Field

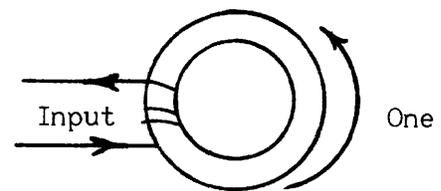


Figure 3B

the core will remain in the state shown above.

To read out of a core, it is necessary to apply a pulse to the probe winding in the direction to produce a binary one. If the core is already in the one state, the output winding will detect no change of field. If however, the core is in a zero state, the change in the magnetic field will cause a current to flow in the output coil as shown in Figure 4.

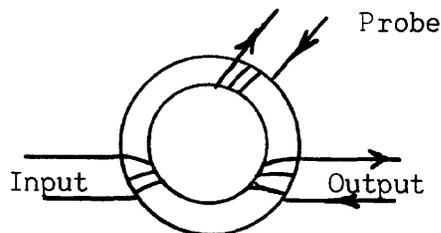


Figure 4

Note however, that if a core is probed, it will always be left in the "one" state irrespective of its initial reading. It is therefore necessary to regenerate the state after reading. This can be accomplished as follows:

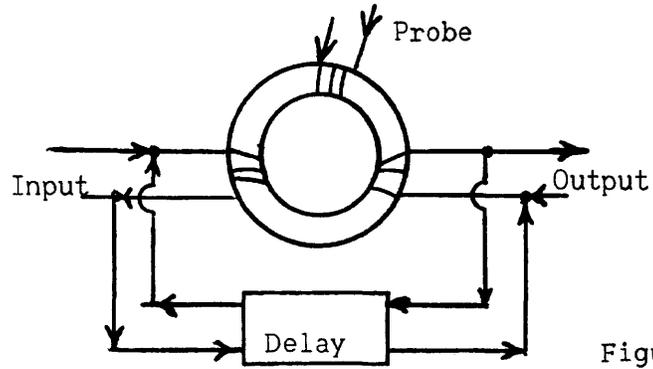


Figure 5

A pulse in the output (representing a zero) is fed, after a suitable delay, to the input winding with the correct phasing to set the core in the zero state. If the core was in the "one" state initially, then no change will be produced by the probe pulse; hence, no pulse will appear at the output coil. Effective access time to the information stored in a core is from 5-10 microseconds.

Cathode Ray Tube

The most common employment of the cathode ray tube, CRT, as a binary memory device is in the Williams type storage. The CRT is basically a small (5-7") television picture tube.

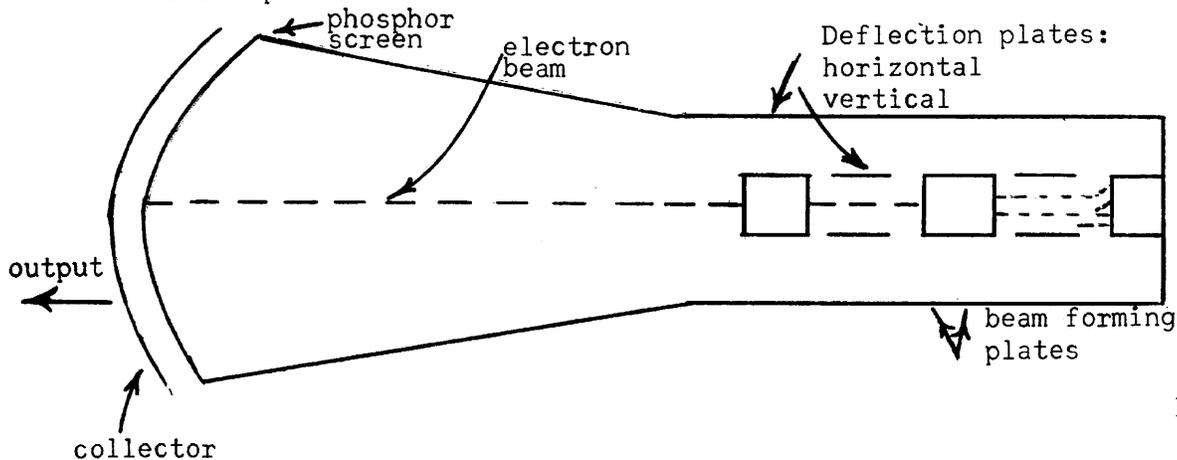


Figure 1

As shown in Figure 1, the CRT contains an electron source which is a special metal filament that emits electrons when heated. As the electrons boil out of the filament, they are formed into a narrow beam (pencil of electrons) by the beam forming plates. This beam may then be made to strike any desired spot on the phosphor screen by suitable voltages on the horizontal and vertical deflection plates. In the television set we are interested in the light emitted when the electron beam strikes the phosphor, but when using the CRT as a storage device we concern ourselves with the voltage produced between the spot on the screen hit by the beam and a collector screen placed just in front of the tube.

When a binary one is to be recorded, the deflection plates are charged to that potential which will direct the electron beam to the desired spot on the screen. The beam is then turned on and sharply focused, placing a dot on the screen. When a zero is recorded, the beam is defocused, putting a blurred spot on the screen. The electronic theory of CRT storage is too involved for this level of discussion, but for our purposes, however, it suffices to say that these two recorded patterns will produce different output voltages between the screen and collector when they are read by directing a defocused beam at the spot.

Since reading is accomplished by recording a zero on the spot, the information read must be rerecorded on the tube as we saw with the magnetic core.

Within a few tenths of a second a spot on the tube will begin to disappear and, thus, the data stored on a tube must be regenerated by reading and re-recording each spot at regular intervals.

It is common practice to store approximately 1000 binary digits per tube. Effective access time to any digit is of the order of 5-10 microseconds.

SECTION III

Dynamic Storage Devices

Magnetic Drum

The following simple experiment is the basis of magnetic drum or magnetic tape storage. A voltmeter is connected to the ends of a coil of wire. When a magnet with north-south poles oriented along the axis of the coil is moved rapidly along this axis, the needle of the voltmeter is deflected indicating that a current has been generated in the coil. If, however, the magnet is oriented at right angles to the coil, no deflection of the needle is observed. We can speak of the magnet orientation in Figure 1A as representing a binary one, while the orientation of 1B represents a binary zero. A succession of

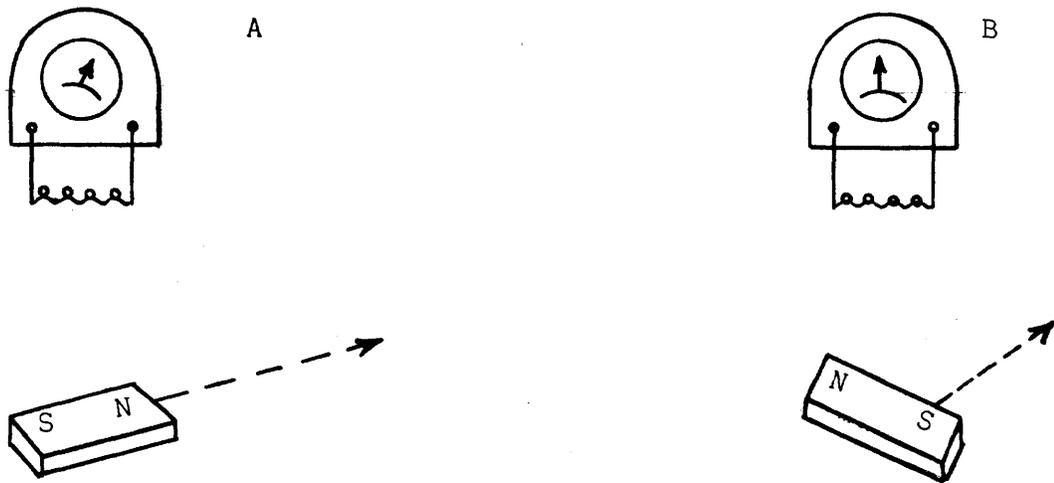


Figure 1

magnetic "spots" in both orientations passing under a coil then will produce a "train" of voltage variations representing the successive orientations of the magnetic spots. Information may be recorded by passing a magnetizable material under this same coil while a voltage train representing the binary information is fed into the coil. A small erase coil, one whose axis is oriented in the direction of the magnet of Figure 1B, is placed in front of the read-write coil and energized only when writing is to be done. This provides the "binary zero" spots.

The storage unit, in practice, consists of an aluminum cylinder coated with iron oxide and rotated by an electric motor past a series of read-write

coils. Each coil may then read or write information on the cylindrical section of the drum immediately under it. Any portion of a drum section is available for reading or writing within one drum revolution. Thus, on the average, an arbitrary spot on a drum section is under the read-write coil and is available for reading or writing within one-half drum revolution. This is called the access time of the drum and is usually in the order of 8-20 milliseconds. The magnetic spots have been recorded at densities as high as 100 binary bits per square inch, and, thus, the magnetic drum is a very cheap memory unit. It does suffer from relatively slow access times, but the drum (as well as the acoustic delay line) can have this access time materially reduced by planning a program in such a way that when information is desired from the drum or is to be written upon it, the proper area of the section is just coming under the read-write coil. This is called minimum latency coding.

Acoustic Delay Line

Consider the situation pictures in Figure 1. When the switch is closed, point A has a voltage level which might represent a binary one, while if the switch is left open A can be said to be at zero voltage level representing binary zero.

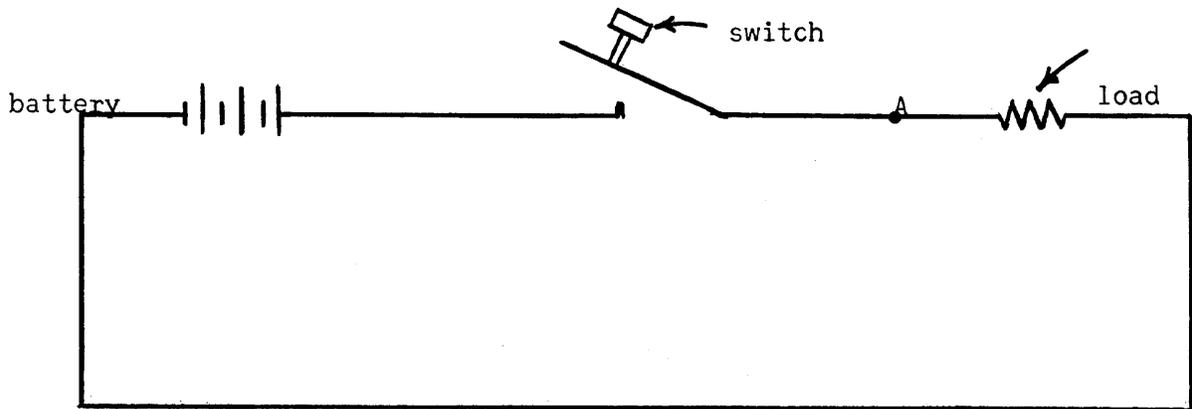


Figure 1

If we operate the switch not oftener than, say, once per second and plot the voltages of A, we would get the following configuration if the number 13 were represented:

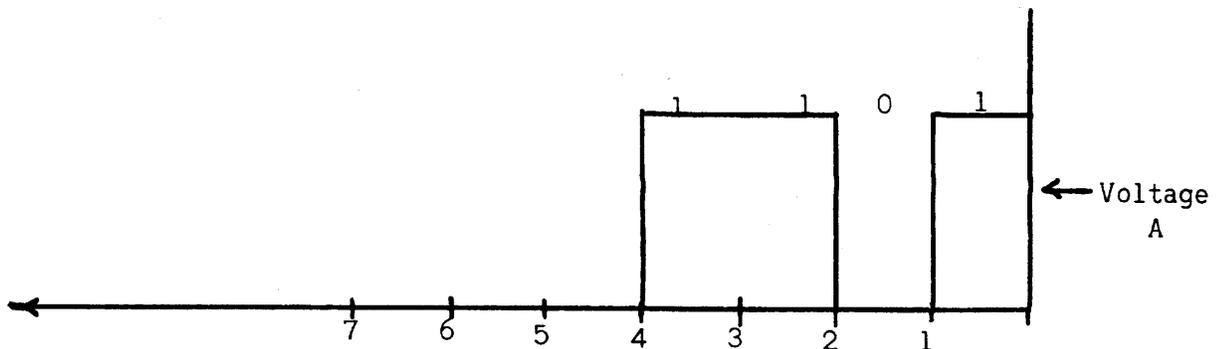


Figure 2

Thus, we can represent binary information by allowing only two possible voltages on a wire. The minimum time between consecutive switch settings fixes the duration of the voltage "pulses" on the line. We call the duration of a pulse the pulse time. In the above case, since the switch setting may be changed once per second, the voltage on the line represents one binary digit for one second. The pulse time is then 1 second. The pulse repetition rate or frequency is the number of binary digits or pulses which can be represented in one second.

Now, suppose we have some kind of "black box" between line A and B as shown in Figure 3 which prevents the voltage on line A from appearing on line B for four seconds. We can plot the voltages of each line as shown in Figure 4. The switch on line A is operated once per second to represent 13.

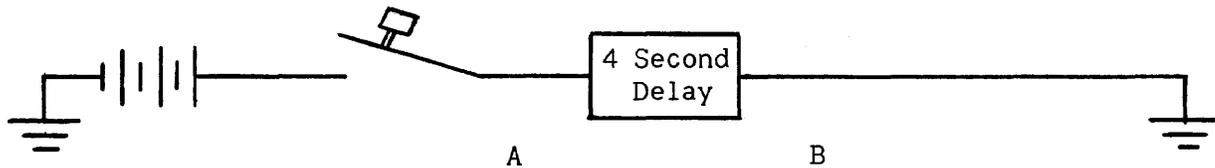


Figure 3

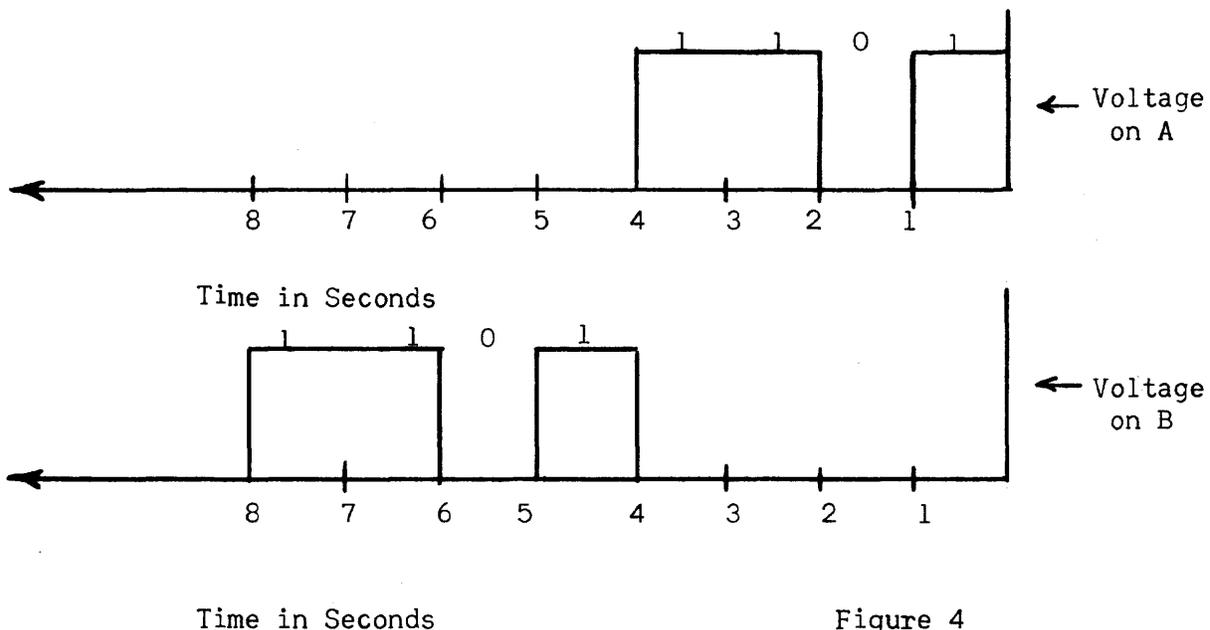


Figure 4

At exactly second 4, if we disconnect the switch and battery from line A and connect A to B, as shown in Figure 5, we have a delay line. Now the voltage

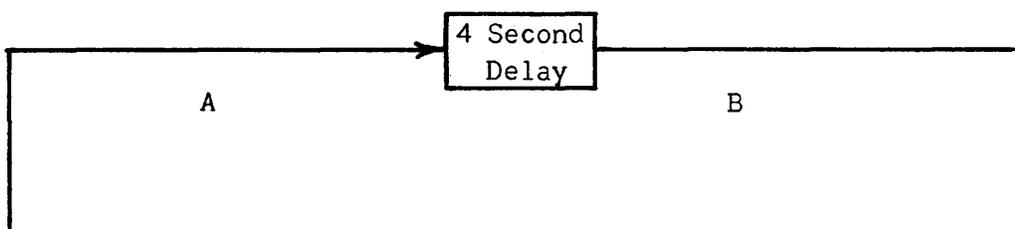


Figure 5

at point A will always be the same as point B. But note that during second 5, line B will have a voltage which existed on line A during second 1; thus, during second 9, line B will again have this same voltage since it is being re-introduced to the four-second delay. This same observation holds for the voltages at B during seconds 6, 7, and 8. They are the same as existed at A during seconds 2, 3, and 4. Further, these same voltages will reappear at B in this same order every four seconds. We can think of this voltage train "circulating" around the closed loop. The presence of the delay preventing the pulses from mixing with one another and losing their identity. Read-out is accomplished by tapping the wire at B. Read-in is somewhat more involved. Through the use of a gate (Chapter 9) placed between A and B, the information emerging from the delay is prevented from re-entering. In its place, the new information is fed in. Of course, it is necessary to put amplifying and pulse shaping devices at the juncture of A and B if the circulation is to be maintained for many cycles.

Next, let us consider the manner by which the 4 second delay in transmitting voltages between A and B is achieved. There are several means for obtaining delays in the transmission of voltages by direct electrical or electromagnetic devices, but usually these methods cannot produce the 40-400 μ s delays required. The method commonly employed to secure this delay is to convert the electric pulses (the voltage train) into sound pulses. Since sound travels more slowly than electricity, it is possible to obtain any desired delay without bulky equipment. After the sound pulses have been sufficiently delayed, they are reconverted to electrical pulses, amplified and shaped and re-introduced to the delay.

The electrical pulses are converted into sound pulses through piezo-electric crystals. This is the same basic type of crystal used in the modern phonograph which converts the record grooves into electrical signals through the vibrations of a needle tracking the grooves. The crystal is cut into the form of a thin slab. When an alternating voltage is applied to two faces of the crystal, it vibrates. Contrariwise, when the crystal is vibrated, an alternating voltage is produced on the crystal faces. Through an electrical process familiar in radio broadcasting, the pulse train is converted to an alternating voltage which in turn is applied to the crystal faces. The alternating voltages representing binary ones then cause the crystal to vibrate and send sound waves into a mercury column. Mercury is often used as the medium in which the sound waves travel because of power and echo considerations. As the soundwaves reach the end of the mercury column, they vibrate a second crystal which then produces an alternating voltage between its faces which is a replica of the input voltages. Figure 6 is a schematic diagram of the process. The access time for information stored in an acoustic delay line depends to a large extent on the number of words stored per line, but typical access times are about 200 μ s.

SECTION 1

Introduction

The purpose of this chapter is to provide an understanding of how elementary electronic building blocks can be used to perform the data manipulation described in Chapter 7. In particular, we have seen that if we can perform addition, shifting and complementing, we are able to add, subtract, multiply and divide numbers expressed in binary form. We shall describe the characteristics of gates, buffers, and flip-flops and show how these units may be interconnected to form adders, complementers, and comparators. Shifting is a somewhat more involved topic and will not be discussed here.* For the sake of simplicity we shall consider dynamic information only; that is, information will be in the form of a voltage train--a positive voltage pulse for a binary one and a zero voltage pulse for a binary zero. When represented in this fashion, the binary number 110101 will appear as shown in Figure 1, the least significant binary columns appearing first in time.

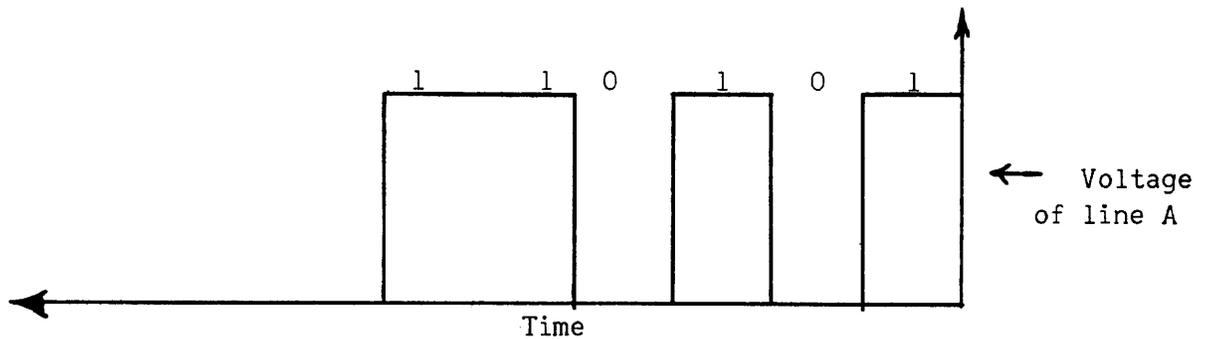


Figure 1

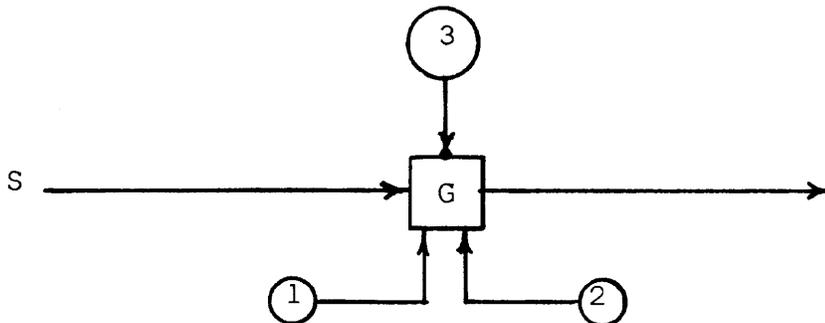
When the voltage of a line is at the binary one level, we speak of the line as carrying a signal.

*See Chapter 9, Univac Programming Manual I

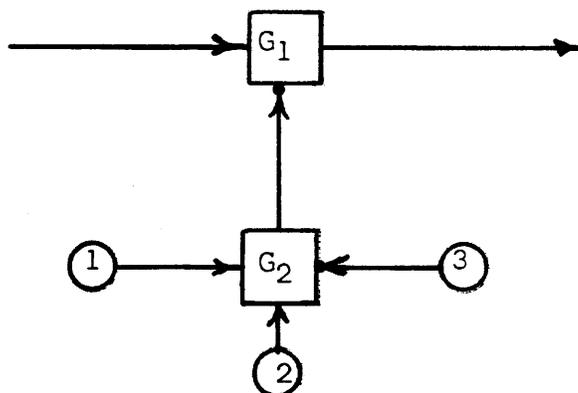
SECTION II

The Logical Building Blocks

The first logical element to be discussed is the gate. Gates form the principal means of switching information from one path to another in computers. As their name indicates, gates permit or prohibit the passage of pulses or signals from one point to another. Gates are indicated by the following symbol:

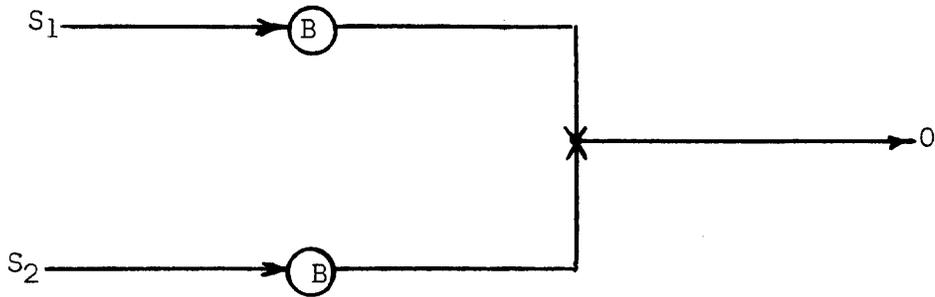


The signal or information train being gated, S , appears on the left. The signals controlling the gate are indicated as 1, 2, and 3. In order for the voltage train, S , to pass through gate G , signals 1 and 2 must be present and signal 3 must be absent. Any other arrangement of signals is sufficient to prohibit S from passing through the gate. Signal 3 is often called an inhibiting signal (note the small circle at the point of connection with the gate which indicates inhibition), while signals 1 and 2 are called permissive signals (without the circle connection). The signals within the large circles always imply the existence of some other devices which generates them. Typically, another gate can generate such signals. In the example shown below, signal S can pass through G_1 if G_2 develops no signal.



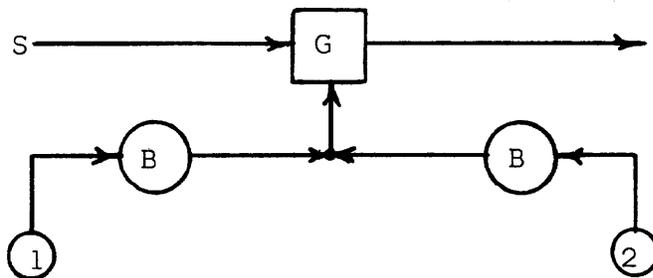
G2 can develop a signal if 1 and 2 are present and 3 absent. Gates are sometimes called "and" circuits because they require the presence of this AND this AND this signal in order to operate. We speak of gates being "open" when all permissive signals are present and all inhibitory signals absent. Gates are said to be "alerted" when some of the required signals are present. It is important to realize that every gate passes ONE signal (or information train) under the influence of OTHER signals. The output of a closed gate is binary zero.

The converse of gating is buffering. The buffer is indicated by a symbol \textcircled{B} . A typical buffering circuit is shown as:



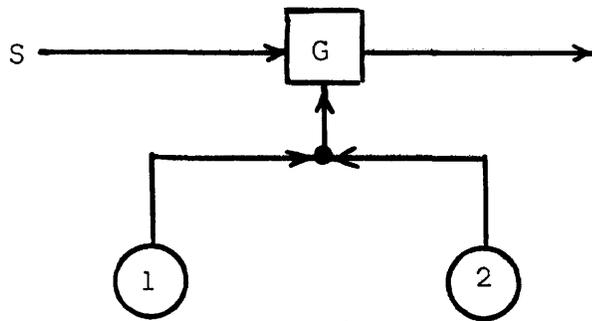
The purpose of buffering is to combine several sources into a single line without interaction among the sources. Thus, signal S_1 cannot pass into S_2 , but only through its buffer B to the output O. Either signal S_1 or S_2 can pass into the output O. For this reason, the buffering circuit is sometimes called an "or" circuit.

Gating signals can be applied through buffers. Thus:

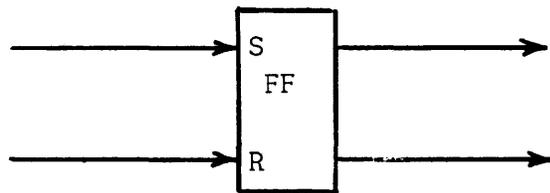


Signal 1 or 2 may open the gate, allowing S to pass through. At least one signal must be present to open the gate: when both signals are present, no new situation has been created.

It is not necessary to show buffering on the logical diagrams if it is understood that such elements exist to prevent back circuits. It will be assumed that no signal can be passed in the reverse direction of the arrows. Thus, the above figure can be redrawn as



In Chapter 8 a number of binary memory devices were discussed. An addition to this list is the flip-flop or trigger pair. Although the flip-flop is seldom used for large memories, it is an extremely versatile memory unit for the control and arithmetic units of a computer. It is indicated in our logical diagrams by the symbol



It is a memory for one binary digit: It has two stable states, one representing zero and the other representing one. These two states are indicated by S (set) and R (reset). Its use is sufficiently broad that the binary notation is not always appropriate. For example, in a binary computer it can store the sign digit; either a + (which could be a one) or a - (which would be a zero). The flip-flop is also useful for converting from a pulse to a static signal. For example, one pulse may indicate when a static signal is to start and another pulse when it is to stop. The flip-flop can be used for generating such a static signal. The duration of the signal is fixed by the interval between pulses.

When a pulse is applied to the "set" input, the flip-flop is said to be set; and when a pulse is applied to the "reset" input, the flip-flop is said to be reset.

If the flip-flop is in the "set" state, the set output line will be at the signal level and the reset output line at the no signal level. The opposite is true for the "reset" condition. In some logical circuits, we shall be interested in only one of the outputs. In this case, only the necessary output will be shown.

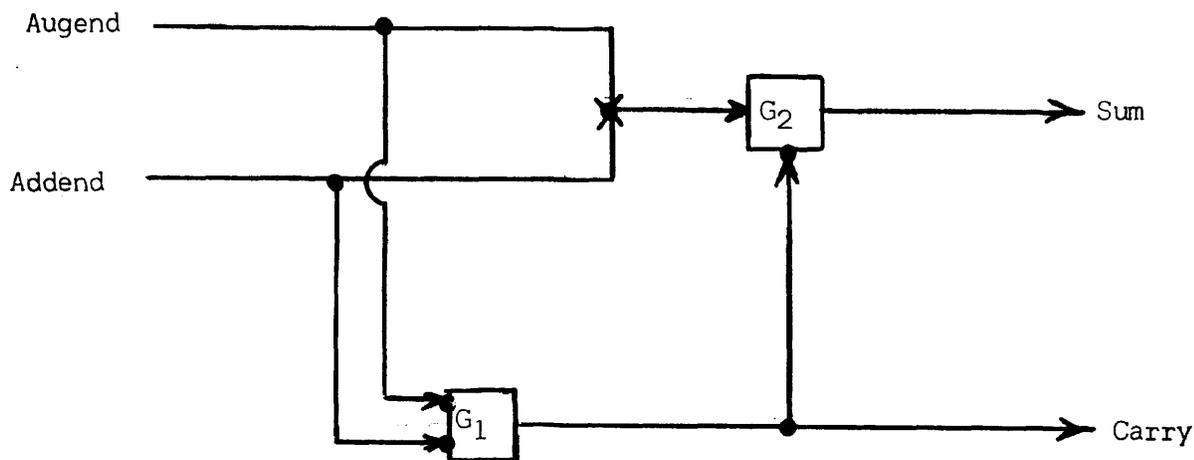
SECTION III

Simple Logical Circuits

The circuit shown below is called a half-adder. This device will add two binary quantities to produce a proper sum and carry:

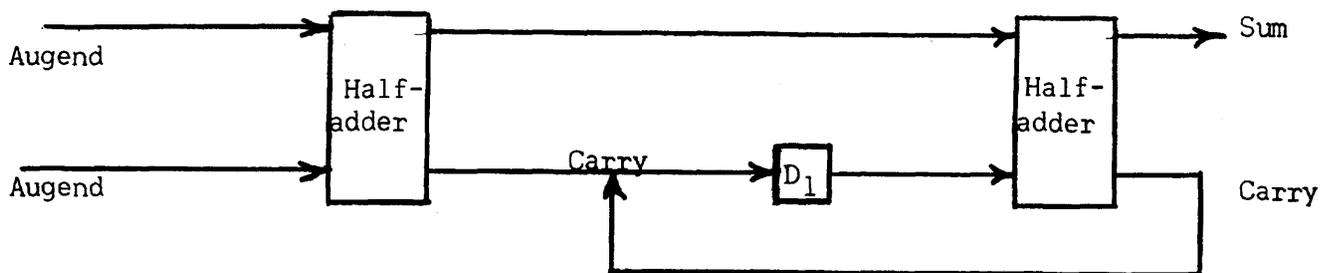
$$\begin{array}{r}
 0 \quad 1 \quad 1 \\
 + 0 \quad + 0 \quad + 1 \\
 \hline
 0 \quad 1 \quad 0 \quad \text{and 1 carry}
 \end{array}$$

The two inputs to the half-adder are called the augend and addend. If no input pulses are present, neither G_1 nor G_2 are open and thus the sum and carry lines are at the binary zero voltage level. If one input is present, G_1 is still closed; but G_2 is open giving a "one" sum and "zero" carry. When both inputs are present



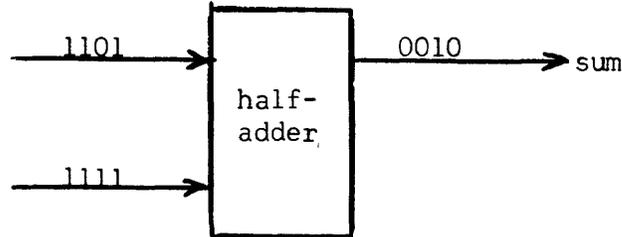
G_1 is open giving a "one" carry and also inhibiting G_2 so that a zero is on the sum line.

By connecting a 1 pulse delay to two half-adders, a full binary adder results:

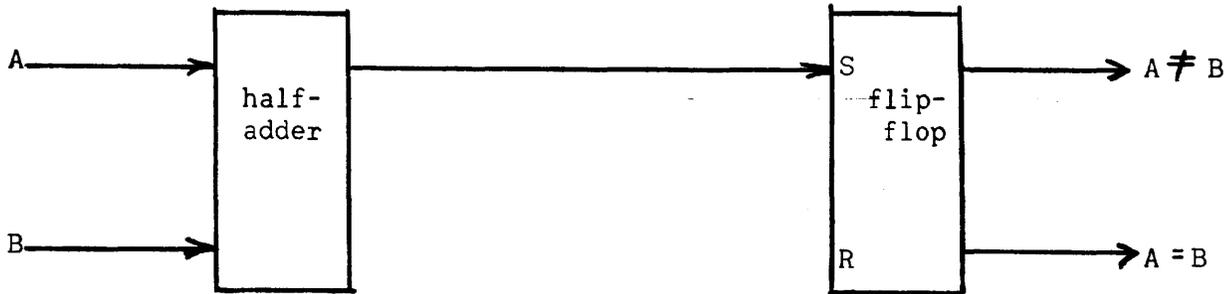


The one pulse delay, D_1 delays the carry from either half-adder until the next binary column is available.

The half-adder has several other important uses besides forming half of a full binary adder. By letting one input to the half-adder consist of a string of ones, the sum output will be the binary complement minus one of the other input. This is easily seen in the following example:

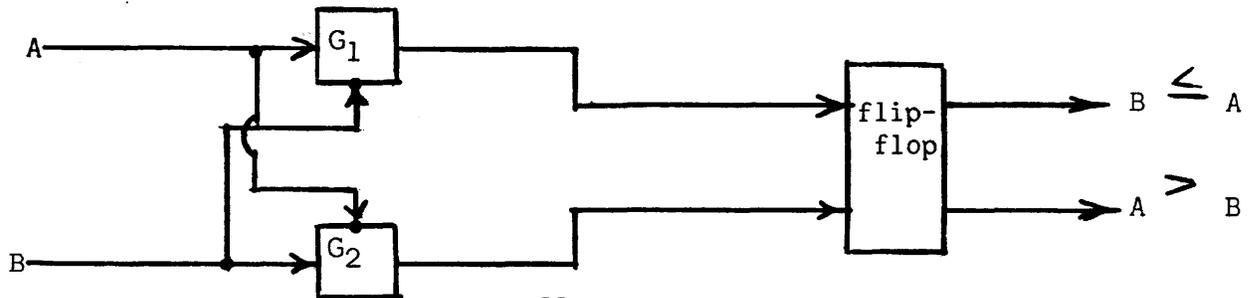


By using a half-adder and flip-flop connected as shown below, we can determine whether two quantities, A and B, are identical. Initially, the flip-flop is



reset. As A and B are fed into the half-adder, pulses will appear on the sum line only if A and B differ in at least one column. Thus, if the flip-flop remains reset after the last binary columns of A and B have passed through HA, a signal is on the reset line indicating $A = B$. If the inputs differed, a sum pulse would have set FF indicating $A \neq B$.

A magnitude comparator is shown below. The flip-flop is initially reset.
Note



that if A and B have an identical binary column, both G_1 and G_2 are closed and the flip-flop remains in its then current state. But if the inputs differ, either G_1 or G_2 will be open (depending on which input has the binary one) and thus a pulse will be sent to either the set or reset side of the FF. After the last binary columns of the inputs have been applied to the gates, the flip-flop will be reset if $A \leq B$ or set if $A > B$.

APPENDIX
SOLUTION TO STUDENT
EXERCISES

CHAPTER 2, SECTION II

Problem 1

```

000  B   100
      A   101
001  A   102
      A   103
002  A   104
      H   105
003  50  105
      90  000
    
```

```

Sum of Receipts → Memory
Print Sum
Stop
    
```

Problem 2

```

000  B   102
      A   103
001  H   104
      A   104
002  A   104
      S   101
003  A   100
      A   100
004  H   104
      50  104
005  90  000
      -----
    
```

```

C + D → Memory
3(C + D) → rA
3(C + D) - B → rA
2A - B + 3(C + D) → rA
    
```

CHAPTER 2, SECTION III

Problem 1

$A \times B = \overset{+}{-} \wedge \text{xx xxx xxx xxx}$

$A \div B = \overset{+}{-} \wedge \text{xx xxx xxx xxx}$

Problem 2

$A \times B = \overset{+}{-} \text{xx xxx xxx x} \wedge \text{xx}$

$A \div B = \overset{+}{-} \text{xx xxx xxx x} \wedge \text{xx}$

Problem 3

$A \times B = \overset{+}{-} \text{xx xxx xxx xx} \wedge \text{x}$

$A \div B = \overset{+}{-} \text{xx xx} \wedge \text{x xxx xxx}$

CHAPTER 2, SECTION IV

Problem 1

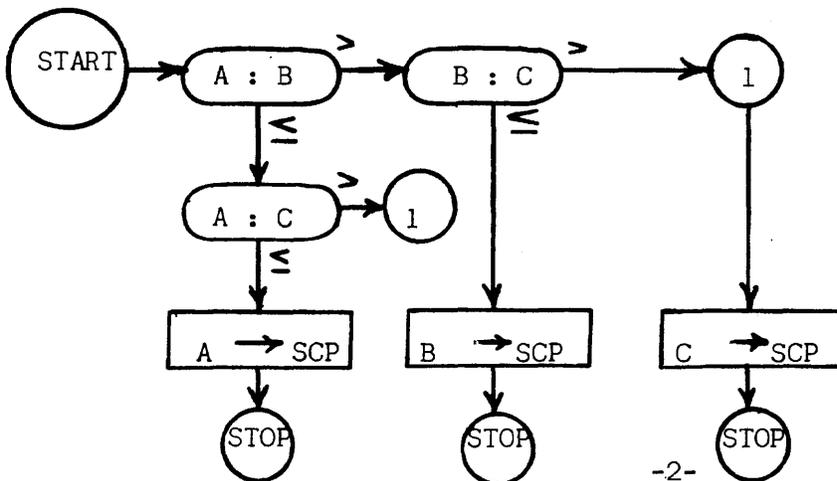
| | | | | | |
|-----|----|-----|----|-----|-----------------------------|
| 000 | B | 100 | | | |
| | | | L | 101 | |
| 001 | 00 | 000 | | | |
| | | | T | 004 | Transfer Control if $A > B$ |
| 002 | L | 102 | | | |
| | | | T | 005 | Transfer Control if $A > C$ |
| 003 | 50 | 100 | | | Print A |
| | | | 90 | 000 | Stop |
| 004 | B | 102 | | | |
| | | | T | 006 | Transfer Control if $C > B$ |
| 005 | 50 | 102 | | | Print C |
| | | | 90 | 000 | Stop |
| 006 | 50 | 101 | | | Print B |
| | | | 90 | 000 | Stop |

Problem 2

| | | | | | |
|-----|-----|-----|-------|-----|---|
| 000 | B | 101 | | | |
| | | | L | 009 | |
| 001 | 00 | 000 | | | |
| | | | Q | 008 | } Transfer Control if $BD = 0$ $BC + BD \rightarrow BC$ |
| 002 | A | 104 | | | |
| | | | H | 104 | |
| 003 | L | 104 | | | |
| | | | B | 102 | |
| 004 | 00 | 000 | | | |
| | | | T | 008 | } Transfer Control if $BS > BC$ $BC - BS \rightarrow BC$ |
| 005 | B | 104 | | | |
| | | | S | 102 | |
| 006 | H | 104 | | | |
| | | | 50 | 100 | Print BN |
| 007 | 50 | 102 | | | Print BS |
| | | | 50 | 103 | Print BA |
| 008 | 90 | 000 | | | Stop |
| | | | ----- | | |
| 009 | 000 | 000 | | | |
| | | | 000 | 000 | |

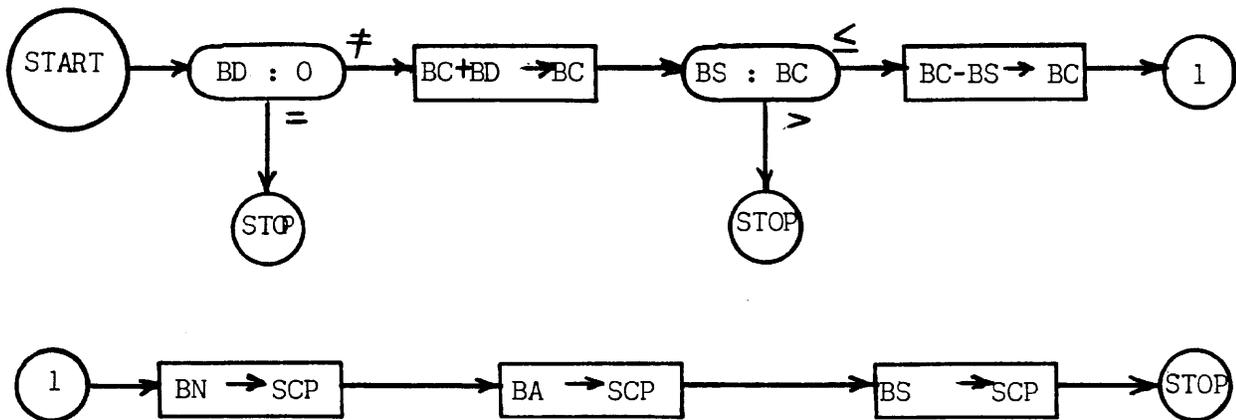
CHAPTER 3, SECTION III

Problem 1

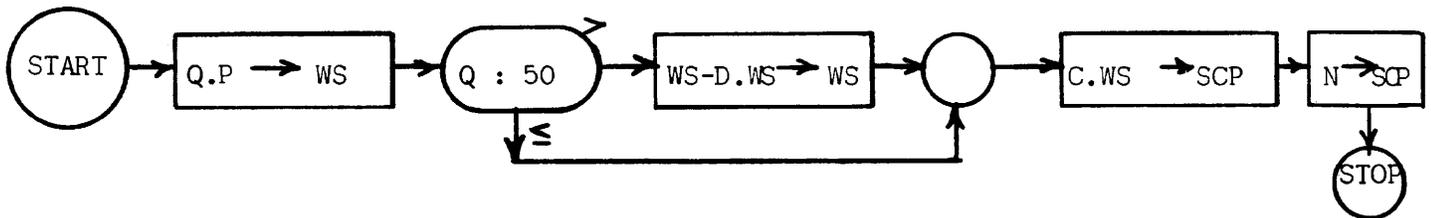


CHAPTER 3, SECTION III

Problem 2



Problem 3

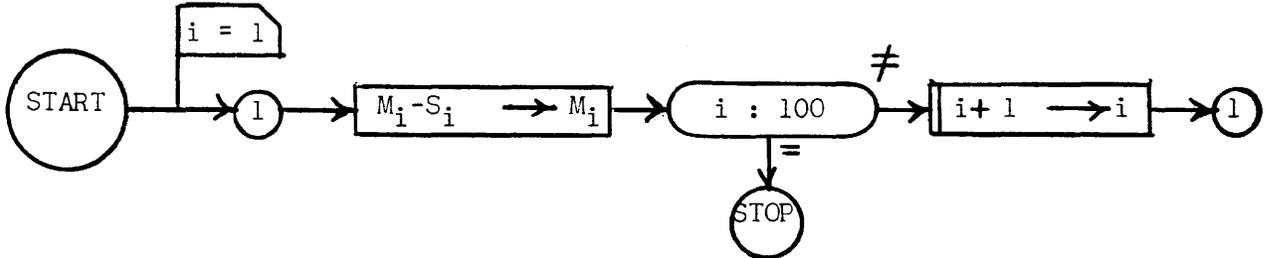


| | | | | |
|-----|-----|-----|-----|-----|
| 000 | L | 100 | | |
| | | | M | 101 |
| 001 | H | 099 | | |
| | | | B | 100 |
| 002 | L | 010 | | |
| | | | T | 006 |
| 003 | L | 099 | | |
| | | | M | 104 |
| 004 | H | 099 | | |
| | | | 50 | 099 |
| 005 | 50 | 103 | | |
| | | | 90 | 000 |
| 006 | L | 099 | | |
| | | | M | 102 |
| 007 | H | 098 | | |
| | | | B | 099 |
| 008 | S | 098 | | |
| | | | H | 099 |
| 009 | 00 | 000 | | |
| | | | U | 003 |
| 010 | 000 | 500 | | |
| | | | 000 | 000 |

Q → rL
 Q.P → WS
 Q → rA
 Transfer Control if Q > 50
 WS → rL
 C.WS → SCP
 N → SCP
 WS → rL
 D.WS → rA
 WS - D.WS → WS

CHAPTER 4, SECTION 11

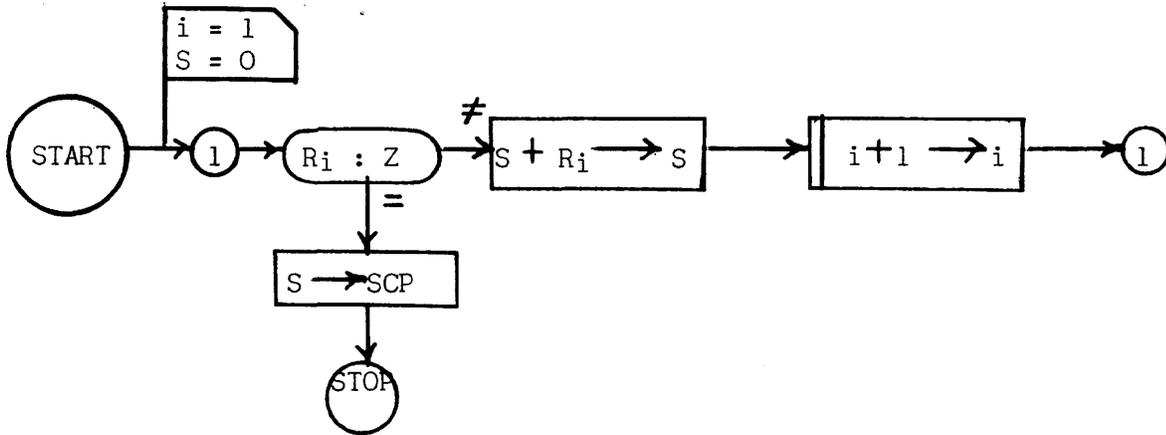
Problem 1



| | | | | | |
|-----|-----|-----|-------|-----|-------------------------------|
| 000 | B | 100 | | | |
| 001 | H | 100 | S | 200 | } $M_i - S_i \rightarrow M_i$ |
| 002 | L | 007 | B | 000 | |
| 003 | A | 008 | Q | 006 | Transfer Control if $i = 100$ |
| 004 | B | 001 | H | 000 | } $i + 1 \rightarrow i$ |
| 005 | H | 001 | A | 009 | |
| 006 | 90 | 000 | U | 000 | |
| | | | ----- | | |
| 007 | B | 199 | S | 299 | |
| 008 | 000 | 001 | 000 | 001 | |
| 009 | 000 | 001 | 000 | 000 | |

CHAPTER 4, SECTION II

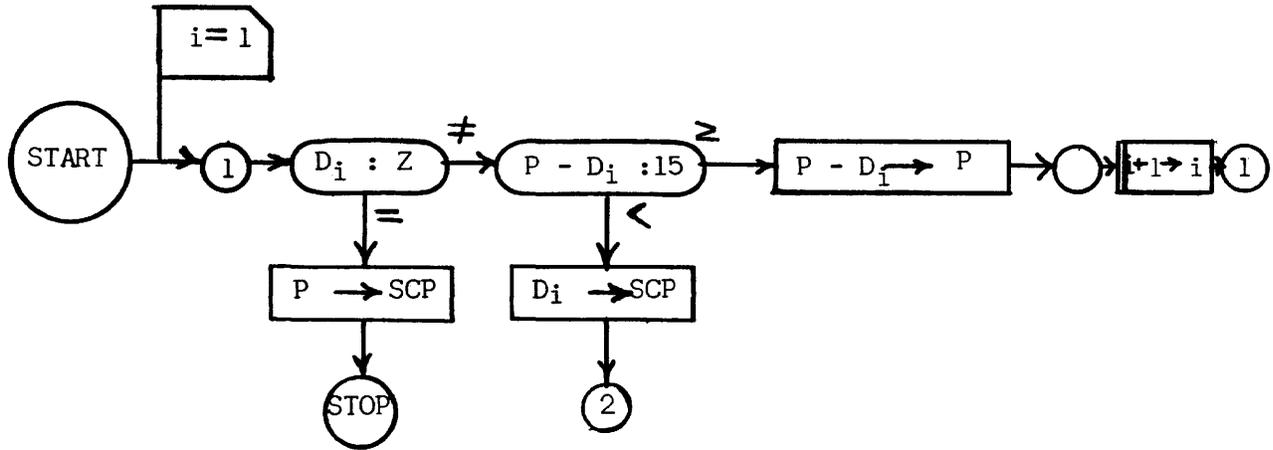
Problem 2



| | | | | | |
|-----|-----|-----|-----|-----|-------------------------------|
| 000 | B | 100 | | | |
| | | | L | 006 | |
| 001 | 00 | 000 | | | |
| | | | Q | 005 | Transfer Control if $R_i = Z$ |
| 002 | A | 007 | | | } $S + R_i \rightarrow S$ |
| | | | H | 007 | |
| 003 | B | 000 | | | } $i + 1 \rightarrow i$ |
| | | | A | 008 | |
| 004 | H | 000 | | | |
| | | | U | 000 | |
| 005 | 50 | 007 | | | Print S |
| | | | 90 | 000 | Stop |
| 006 | ZZZ | ZZZ | | | |
| | | | ZZZ | ZZZ | |
| 007 | 000 | 000 | | | |
| | | | 000 | 000 | } -S |
| 008 | 000 | 001 | | | |
| | | | 000 | 000 | |

CHAPTER 4, SECTION II

Problem 3



| | | | | |
|-----|-----|-----|-----|-----|
| 000 | B | 100 | | |
| | | | L | 009 |
| 001 | H | 098 | | |
| | | | Q | 008 |
| 002 | B | 099 | | |
| | | | S | 098 |
| 003 | L | 010 | | |
| | | | T | 007 |
| 004 | 50 | 098 | | |
| | | | 00 | 000 |
| 005 | B | 000 | | |
| | | | A | 011 |
| 006 | H | 000 | | |
| | | | U | 000 |
| 007 | H | 099 | | |
| | | | U | 005 |
| 008 | 50 | 099 | | |
| | | | 90 | 000 |
| 009 | ZZZ | ZZZ | | |
| | | | ZZZ | ZZZ |
| 010 | 000 | 000 | | |
| | | | 001 | 499 |
| 011 | 000 | 001 | | |
| | | | 000 | 000 |

Transfer Control if $D_i = Z$

Transfer Control if $P - D_i \geq 15$

Print D_i

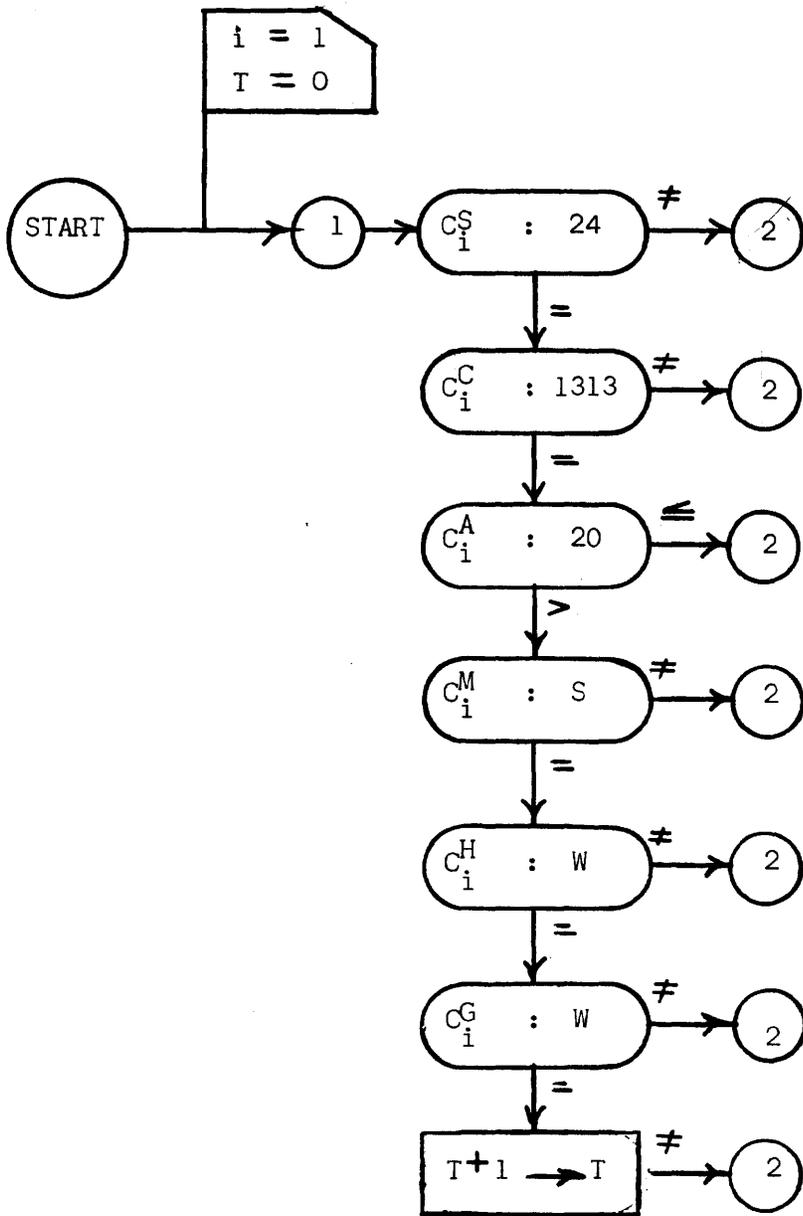
$i + 1 \rightarrow i$

$P - D_i \rightarrow P$

Print P

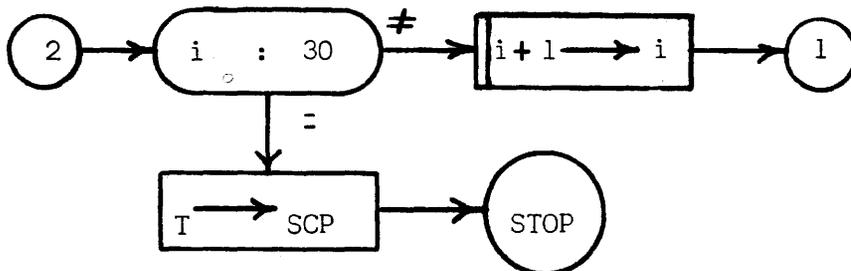
Stop

Problem I
 Chapter V
 Section III



C_i = ith census item

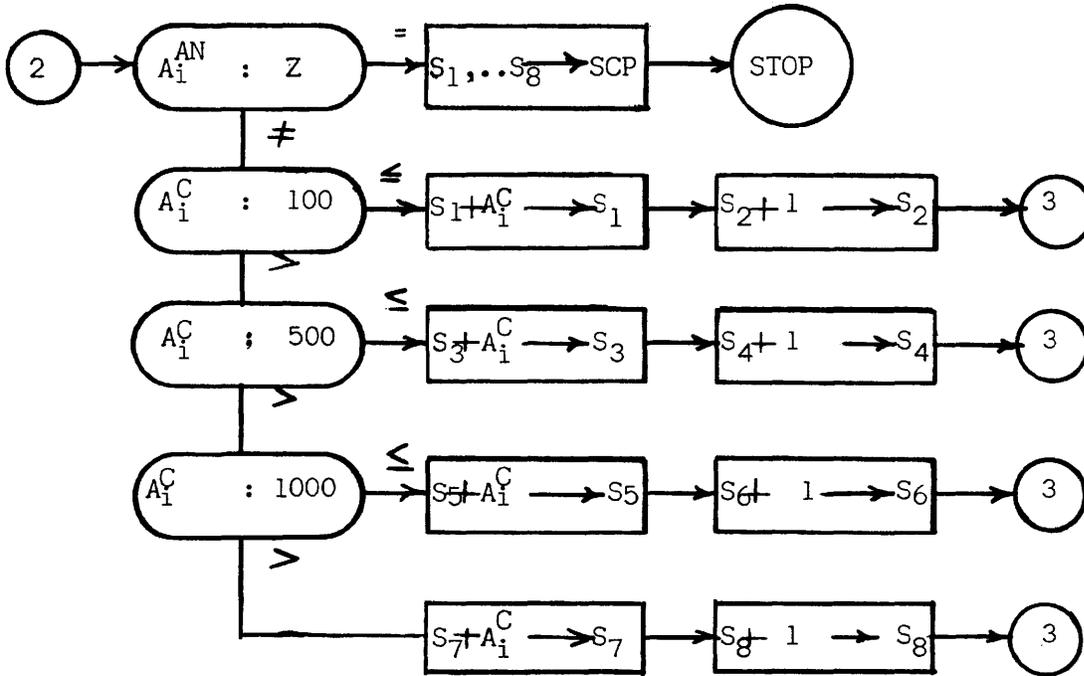
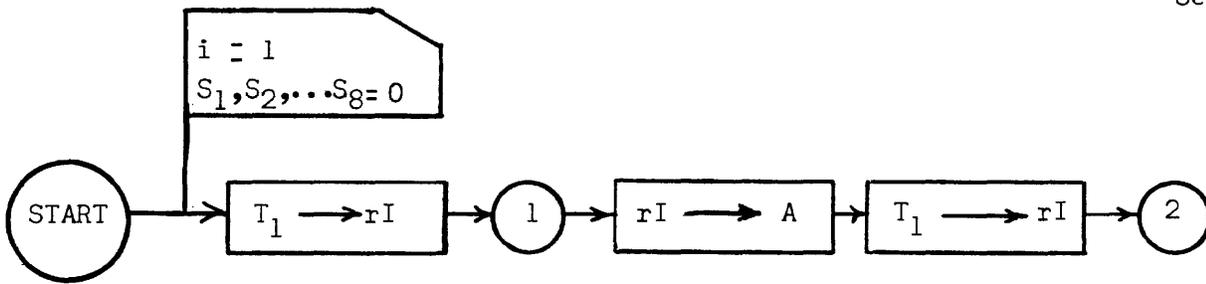
The superscripts represent the various item fields. For example C_i^S = the state codes of the ith item.



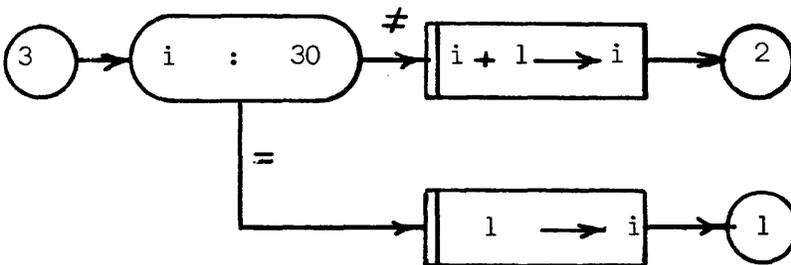
CHAPTER 5, SECTION III

Problem 1

| | | | | | |
|-----|-----|-----|-----|-----|---|
| 000 | B | 100 | | | |
| | | | L | 016 | |
| 001 | 00 | 000 | Q | 008 | Transfer Control if $C_i^s = 24, C_i^f = 1313$ |
| 002 | B | 005 | L | 017 | |
| 003 | 00 | 000 | Q | 007 | Transfer Control if $i = 30$ |
| 004 | A | 018 | H | 005 | } $i + 1 \longrightarrow i$ |
| 005 | V | 100 | W | 100 | |
| 006 | 00 | 000 | U | 000 | |
| 007 | 50 | 019 | 90 | 000 | Print T STOP |
| 008 | B | 101 | L | 020 | |
| 009 | 00 | 000 | T | 011 | Transfer Control if $C_i^A > 20$ |
| 010 | 00 | 000 | U | 002 | |
| 011 | 03 | 000 | L | 021 | |
| 012 | 00 | 000 | Q | 014 | Transfer Control if $C_i^m = S, C_i^h = W, C_i^g = W$ |
| 013 | 00 | 000 | U | 002 | } $T + 1 \longrightarrow T$ |
| 014 | B | 019 | A | 022 | |
| 015 | H | 019 | U | 002 | |
| 016 | 024 | 000 | 131 | 300 | |
| 017 | V | 158 | W | 100 | |
| 018 | 000 | 002 | 000 | 000 | } T |
| 019 | 000 | 000 | 000 | 000 | |
| 020 | 002 | 020 | 000 | 000 | |
| 021 | 005 | 0W0 | 00W | 000 | |
| 022 | 000 | 000 | 000 | 001 | |



A = A block of meter items from Tape 1
 A_i = ith meter item of Block A
 A_i^{AN} = account number of A_i
 A_i^C = consumption of A_i



Problem 1

| | | | | |
|-----|----|-----|----|-----|
| 000 | 11 | 000 | | |
| 001 | 31 | 100 | 00 | 000 |
| 002 | B | 100 | 00 | 000 |
| 003 | 00 | 000 | L | 044 |
| 004 | B | 101 | Q | 027 |
| 005 | 00 | 000 | L | 045 |
| 006 | A | 033 | T | 016 |
| 007 | B | 034 | H | 033 |
| 008 | H | 034 | A | 048 |
| 009 | B | 012 | 00 | 000 |
| 010 | 00 | 000 | L | 049 |
| 011 | A | 050 | Q | 014 |
| 012 | V | 100 | H | 012 |
| 013 | 00 | 000 | W | 100 |
| 014 | B | 051 | U | 002 |
| 015 | 00 | 000 | H | 012 |
| 016 | L | 046 | U | 001 |
| 017 | A | 036 | T | 020 |
| 018 | B | 037 | H | 036 |
| 019 | H | 037 | A | 048 |
| 020 | L | 047 | U | 009 |
| 021 | A | 039 | T | 024 |
| 022 | B | 040 | H | 039 |
| 023 | H | 040 | A | 048 |
| 024 | A | 042 | U | 009 |
| 025 | B | 043 | H | 042 |
| 026 | H | 043 | A | 048 |
| 027 | 50 | 032 | U | 009 |
| | | | B | 027 |

$T_1 \longrightarrow rI$
 $rI \longrightarrow A, T_1 \longrightarrow rI$

Transfer Control if $A_i^{an} = Z$

Transfer Control if $A_i^c > 100$
 $S_1 + A_i^c \longrightarrow S_1$
 $S_2 + 1 \longrightarrow S_2$

Transfer Control if $i = 30$
 $i + 1 \longrightarrow i$

$i \longrightarrow i$

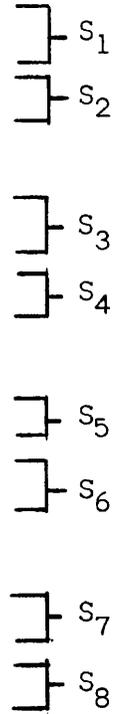
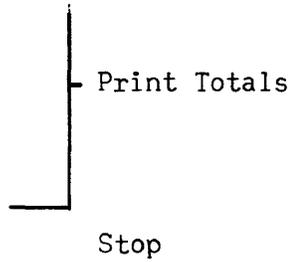
Transfer Control if $A_i^c > 500$
 $S_3 + A_i^c \longrightarrow S_3$
 $S_4 + 1 \longrightarrow S_4$

Transfer Control if $A_i^c > 1000$
 $S_5 + A_i^c \longrightarrow S_5$
 $S_6 + 1 \longrightarrow S_6$

$S_7 + A_i^c \longrightarrow S_7$
 $S_8 + 1 \longrightarrow S_8$

Problem 1 (cont.)

| | | | | |
|-----|-----|-----|-----|-----|
| 028 | L | 052 | | |
| | | | Q | 031 |
| 029 | A | 053 | | |
| | | | H | 027 |
| 030 | 00 | 000 | | |
| | | | U | 027 |
| 031 | 90 | 000 | | |
| | | | 00 | 000 |
| 032 | rΔΔ | 01- | | |
| | | | Δ10 | 0ΔΔ |
| 033 | 000 | 000 | | |
| | | | 000 | 000 |
| 034 | ΔΔ0 | 000 | | |
| | | | 000 | 000 |
| 035 | rΔL | 01- | | |
| | | | Δ50 | 0ΔΔ |
| 036 | 000 | 000 | | |
| | | | 000 | 000 |
| 037 | ΔΔ0 | 000 | | |
| | | | 000 | 000 |
| 038 | rΔ5 | 01- | | |
| | | | 100 | 0ΔΔ |
| 039 | 000 | 000 | | |
| | | | 000 | 000 |
| 040 | ΔΔ0 | 000 | | |
| | | | 000 | 000 |
| 041 | r10 | 01- | | |
| | | | OVE | RΔΔ |
| 042 | 000 | 000 | | |
| | | | 000 | 000 |
| 043 | ΔΔ0 | 000 | | |
| | | | 000 | 000 |
| 044 | ZZZ | ZZZ | | |
| | | | ZZZ | ZZZ |
| 045 | 000 | 000 | | |
| | | | 000 | 100 |
| 046 | 000 | 000 | | |
| | | | 000 | 500 |
| 047 | 000 | 000 | | |
| | | | 001 | 000 |
| 048 | 000 | 000 | | |
| | | | 000 | 001 |
| 049 | V00 | 158 | | |
| | | | W00 | 100 |
| 050 | 000 | 002 | | |
| | | | 000 | 000 |
| 051 | V00 | 100 | | |
| | | | W00 | 100 |
| 052 | 500 | 043 | | |
| | | | B00 | 027 |
| 053 | 000 | 001 | | |
| | | | 000 | 000 |



DENOMINATED PAYROLL

DEMONSTRATION

Operating Instruction:

1. Set SC printer on normal, right margin to extreme right. Paper size should permit typing at least 62 characters.
2. Mount instruction tape on Uniservo 1, and do initial read for that servo.
3. Routine will print headings and stop on a type-in.
4. Type in pay to be denominated in the following format:
 000 XXX XX0 000
↑ (assumed decimal point)
5. Routine will print appropriate denominations and stop on the type-in of Step 3. Type in next pay to be denominated.
6. When the last pay is to be denominated, set break point 1 and then type in the last pay.
7. This pay will be denominated and computer will stop on a Q1. Force no transfer. Computer will print a total heading, the total pay, and the total number of denominated bills. Tape 1 will rewind and computer will stop.

| | | | | | |
|-----|----|-----|----|-----|--------------------------------------|
| 000 | 11 | 000 | | | |
| | | | 30 | 060 | } |
| 001 | 50 | 040 | 50 | 041 | |
| 002 | 50 | 042 | 50 | 043 | |
| 003 | 50 | 044 | 50 | 045 | |
| 004 | 10 | 100 | B | 101 | |
| 005 | A | 100 | C | 101 | } |
| 006 | B | 100 | L | 046 | |
| 007 | Q | 000 | T | 017 | Transfer Control if $P \geq D_i$ |
| 008 | B | 102 | 00 | 000 | W → rA |
| 009 | R | 039 | U | 033 | Transfer control to edit W and print |

| | | | | | |
|-----|----|-------|---------|---|---|
| 010 | 00 | 000 | | | |
| 011 | A | 110 | B 102 | } | $T_i + W \longrightarrow T_i$ |
| 012 | | C 102 | C 110 | | |
| 013 | L | 056 | B 006 | | |
| 014 | A | 057 | Q 021 | | Transfer Control if $i = 10$ |
| 015 | B | 011 | C 006 | } | $i + 1 \longrightarrow i$ |
| 016 | | | A 058 | | |
| 017 | J | 103 | U 006 | | |
| 018 | C | 100 | S 103 | | $P - D_i \longrightarrow P$ |
| 019 | A | 057 | B 102 | | $W + 1 \longrightarrow W$ |
| 020 | 00 | 000 | C 102 | | |
| 021 | 00 | 000 | U 006 | | |
| 022 | 50 | 059 | Q1 030 | | Force no transfer to print totals |
| 023 | 50 | 067 | 50 101 | } | Heading for totals, $S, . \longrightarrow SCP$ |
| 024 | | | 000 000 | | |
| 025 | B | 110 | 00 000 | | $T_i \longrightarrow rA$ |
| 026 | R | 039 | U 033 | | Transfer Control to edit T_i and print |
| 027 | B | 024 | L 060 | | |
| 028 | 00 | 000 | Q 066 | } | Transfer Control if $i = 10$ $i + 1 \longrightarrow i$ |
| 029 | A | 058 | C 024 | | |
| 030 | 00 | 000 | U 024 | | |
| 031 | B | 061 | C 006 | } | $l \longrightarrow i$ |
| 032 | | | 50 | | |
| 033 | H | 104 | U 004 | | /// /// /// /// $\longrightarrow SCP$ |
| 034 | B | 064 | K 000 | | $rA \longrightarrow WS$ |
| | | | 00 000 | | |

| | | | | |
|-----|----------------|----------------|----------------|----------------|
| 035 | .1 | 000 | T | 035 |
| 036 | C | 105 | F | 105 |
| 037 | B | 065 | E | 104 |
| 038 | C | 104 | 50 | 104 |
| 039 | { | 000 000 | 000 | 000 } |
| 040 | AAA | APA | YAA | AAA |
| 041 | AA2 | 0.A | A10 | .AA |
| 042 | A5. | AAA | 2.A | AA1 |
| 043 | .AA | .50 | AA. | 25A |
| 044 | A.1 | 0AA | .05 | AA. |
| 045 | 01A | AA | AA | AA |
| 046 | 000 | 020 | 000 | 000 |
| 047 | 000 | 010 | 000 | 000 |
| 048 | 000 | 005 | 000 | 000 |
| 049 | 000 | 002 | 000 | 000 |
| 050 | 000 | 001 | 000 | 000 |
| 051 | 000 | 000 | 000 | 000 |
| 052 | 000 | 000 | 500 | 000 |
| 053 | 000 | 000 | 250 | 000 |
| 054 | 000 | 000 | 100 | 000 |
| 055 | 000 | 000 | 050 | 000 |
| 056 | B00 | 100 | 010 | 000 |
| 057 | 000 | 000 | L00 | 055 |
| 058 | 000 | 001 | 000 | 001 |
| 059 | RT | OTA | 000 | 001 |
| | | | LS- | AA |

~~AAA~~ ~~AAA~~ ~~AAA~~ ~~AA~~ ~~AA~~ - → rA

Edited number → SCP

Preset return

Denominations

| | | | | | |
|-----|----------------|----------------|----------------|----------------|------|
| 060 | B00 | 119 | | | |
| | | | 000 | 000 | |
| 061 | B00 | 100 | | | |
| | | | L00 | 046 | |
| 062 | A00 | 110 | | | |
| | | | C00 | 110 | |
| 063 | xxx | xxx | | | |
| | | | xxx | xxx | |
| 064 | 1-- | --- | | | |
| | | | --- | --- | |
| 065 | xxx | xxx | | | |
| | | | xx | xx- | |
| 066 | 81 | 000 | | | Stop |
| | | | 90 | 000 | |
| 067 | .xx | xxx | | | |
| | | | xxx | xxx | |

Fill with zeros

