DECUS
PROGRAM LIBRARY

DECUS NO.	8-231
TITLE	DATA PROCESSING ON THE PDP-8/S
AUTHOR	Frederick W. Holzwarth
COMPANY	George Washington High School Philadelphia, Pennsylvania
DATE	November 15, 1969
SOURCE LANGUAGE	

This unit was used for a 5 period per week, one year course at George Washington High School, Phila., Penna. Approximately one period per week was devoted to demonstrations at the computer. No class time was scheduled for "hands on " use by students, but the computer was available for many periods per week and all students got "hands on" experience during lunch periods, before, and after school.

Hardware included a Digital Equipment Corporation PDP8/S with high speed reader and punch, an on-line ASR33, and an off-line ASR33. Languages used were FOCAL, FORTRAN, PAL III, AND MACRO-8. Other software included editing and debugging programs.

TABLE OF CONTENTS

TABLE OF CONTENTS
(continued)

0.1 Brief History of Computers

Did you know that digital computers have been in existance for many hundreds of years? Not electronic digital calculators, which is what we understand today by the word computer. There is the abacus with which some of you are familiar. Your fingers and your toes may be used. Do not laugh, for there is system Russian Peasant Multiplication, by means of which the fingers are used for multiplication of numbers larger than ten. The first mechanical calculators were developed in the 17th century. One of these called Pascal's Calculator after its inventor, is sold today in plastic for less than $5.00.

Although Pascal s invention was improved down through the years it was not until the nineteenth century that a calculator was invented that could operate automatically. Although it was never built, it was the first to include many of the principles of modern computers, including memory. It did not need human operation at every step. This was Babbages "analytical engine."

In 1879, Herman Hollerith, a 19 year old employee of the U:S: Census Office, began work on an Electronic Tabulating System. Using some of Babbages ideas and the work of the Frenchman, Joseph Jacuard, who had invented punched paper cards, his "Census Machine" was finished in time to be used in the Census of 1890. Hollerith's name is preserved today in the language of computers, having been given to alphabetical data. Hollerith s machine, as well as all desk adding machines and calculators(until recently) was based on electro-mechanical principles. That is, it was built from mechanical gears, cams, levers, springs, etc.(like your car's engine) and electrical switches and relays(like the light switches in your house).

The first electronic, fully automatic, computer was built between 1943 and 1946 at the University of Pennsylvania's Moore School of Electrical Engineering by Drs. Eckert and Mauchly. It was called ENIAC and wqs designed to compute ballistic tables for the ARMY. It used vacuum tubes and was much faster and more automated than any of its electro-mechanical predecessors. It weighed 30 tons, took up the space of 80 living rooms, and included 18,000 tubes and 500,000 connectors. However, it was 1000 times faster than any calculator in existence at that time. Incidentally, Mauchly and Eckert went on to find the UNIVAC division of Sperry Rand, the second largest computer manufacturer today. This was the first generation of computers.

The concept of stored in memory programs and the technical advance of transistors was the second generation of computers in the 1950s. The third generation of computers in the 1960s was marked by the concept of shared time and the technical advance of integrated circuits. We are now entering the forth generation of computers, but it is too early to tell what advances will characterize it.

In addition to digital computers there are ANALOG Computers, which operate on different principles. What separates a computer from a desk calculator? Basically there are only four functions that only a computer can do. These are high speed, repeated calculations, memory, and decisions. It must be admitted that today the line between computers and calculators is beginning to blur and some recent desk calculators can do one or two of these as well as some recent "small" computers. However, only a computer can do all of them.

DATA PROCESSING

1. Binary arithmetic

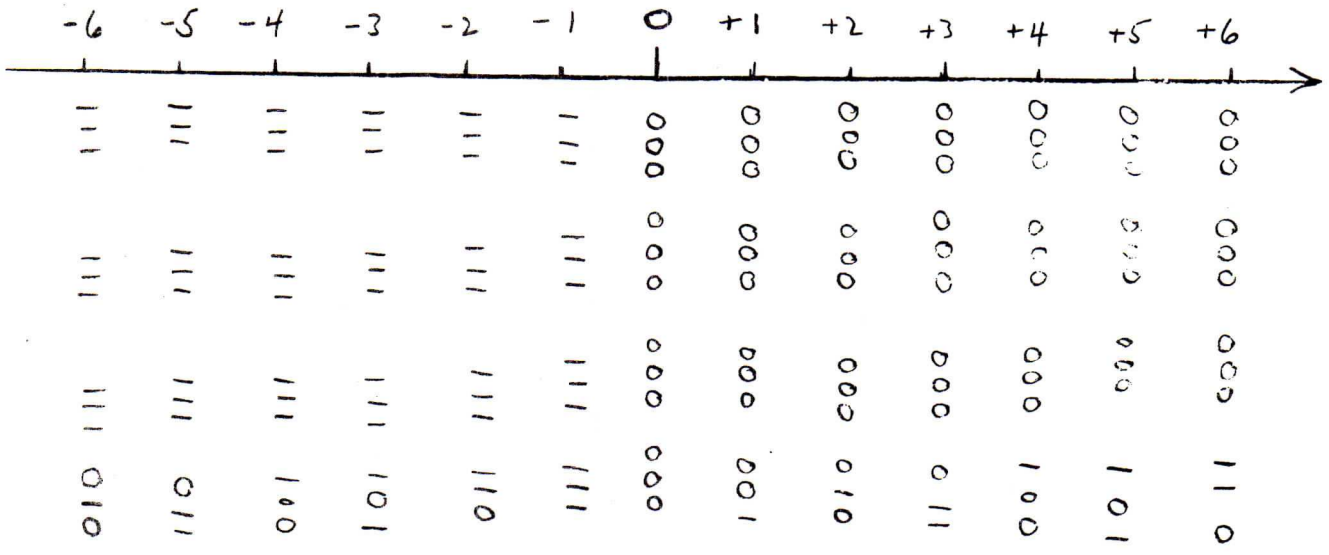
a. Each column, starting on the right, represents a power of 2. $2^{10}..2^4 2^3 2^2 2^1 2^0$. In each column a 1 or 0 is entered. The 1 or 0 is called a bit. The PDP8 uses 12 bit numbers called words.
Example: 000 000 000 110 = $1*2^2 + 1*2^1 + 0*2^0 = 6$

b. To add, we carry as in the decimal system. The only combinations used are: 1 + 0 = 1 , 1 + 1 = 0.

Example:
```
  000 000 000 110
+ 000 000 000 011
  000 000 001 001
```

c. To indicate the sign (+ or -) in 2s complement arithmetic we use the left most bit. But oniy 11 bits are used for the binary number. To form an additive inverse (negative) we change all 0s to 1, all 1s to 0, and add 1. This is also called the 2s complement of the original number. **Leftmost bit indicates + if 0, and - if 1.**

Example: 000 000 000 101 = +5
111 111 111 011 = -5



d. To subtract, we use 2s complement addition. We write the additive inverse of the subtrahend and then add the minuend and the additive inverse of the subtrahend.

Example: Subtract +3 from +6
A +6 = 000 000 000 110
+3 = 000 000 000 011
B -3 = 111 111 111 101 (additive inverse)

Now add A to B and you will get 1 000 000 000 011. This consists of 13 bits. The left most bit overflows and is lost. The correct answer is 000 000 000 011,

which of course = +3.

Example: Subtract -4 from +7

A + 7 = 000 000 000 111

-4 = 111 111 111 100

B-(-4) = 000 000 000 100

Add A to B and get 000 000 001 011 = +11

e. Exercises

1. Convert the following to 12 bit binary words:
   a) +64
   b) -8
   c) -15

2. Convert the following 12 bit binary words to decimal numbers (incl sign):
   a) 000 000 010 000
   b) 000 000 111 001
   c) 111 111 100 101
   d) 110 111 111 111

3. What is the largest decimal number we can represent in a 12 bit binary word, reserving the left most bit for a sign indicator?

4. What is the largest decimal number we can represent in a 12 bit binary word, with no sign indicator?

5. Do the following problems in 2s complement arithmetic:
   a) (+7) + (+3) =
   b) (+4) - (+6) =
   c) (-9) + (-2) =
   d) (-3) - (-8) =

2. The octal number representation of a binary number

a. In the octal base system each column, starting on the right, represents a power of 8. 8^3 8^2 8^1 8^0. We will use only 4 colmns. In each column we may enter a number from 0 to 7.

Example $0021_8 = 0 * 8^3 + 0 * 8^2 + 2 * 8^1 + 1 * 8^0 = 17_{10}$

b. To add, we carry as in the decimal system. The combinations are the same as in the decimal, except that 8 and 9 are not used. Any sum>7 results in a carry. Subtract 8 from the sum and write the difference in the column being added.

Examples:

```
  0007        0174
+0006       +0127
 0015        0323
```

(7 + 6 = 13. Carry 1. 13 - 8 = 5)

c. To subtract, we borrow 8 (not 10 as in the decimal system) and reduce the number in the column from which we borrowed by 1. All combinations are the same as decimal, except that 8 and 9 are excluded.

Examples:

0015	2376	2121
-0007	- 2311	-2076
0006	0065	0023

(8 + 5 = 13. 13 - 7 = 6)

d. It is inconvenient to write numbers and addresses like 101 110 001 110. We use the octal system as a shorthand. The computer does not do arithmetic in the octal system nor do we enter any octal numbers into the computer without converting to binary

Table of Conversion

Octal	Binary
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

e. Grouping our 12 bit binary word into 4 groups of 3 bits each, we can represent each group of 3 bits by a single octal digit from 0 to 7.

Example: $0\ 0\ 2\ 1_8 = 17_{10}$

$000\ 000\ 010\ 001_2 = 17_{10}$

f. Although octal arithmetic is not used by the computer, we use octal counting and arithmetic. Remember that in counting the first number after 7 is 10, after 17 is 20, after 47 is 50, etc.

Example: How many addresses are there from 2311_8 to 2376_8 inclusive?
Ans. Subtract (see c. above) to get the answer 0065_8. This means that there are 53_{10} addresses available.

g. Exercises

1. Convert the following from binary to octal:
   a) 100 000 101 000
   b) 001 101 110 001
   c) 110 111 010 000
   d) 101 001 001 100

2. Convert the following from octal to binary
   a) 2071
   b) 4123
   c) 6572
   d) 1604

3. How many addresses are there from address 0025_8 to address 0035_8 ? Give answer as decimal number. INCLUSIVE

4. How many addresses are there from address 2311_8 to address 3277_8 ? INCLUSIVE

5. If we add 15_{10} addresses to the address 0056_8, what is the last (octal) address?

6. If we add 45_{10} addresses to the address 1254_8, what is the last (octal) address?

3. Storage and retrieval of information

a. Fig. 1 below is a simplified version of the PDP8 console. Suppose we wish to load the number 1025_8 into the computer for future use. It will go into the memory at the address you select. Remember that you must select the address. If you forget, the computer will store your number at an address, but you will be unable to use the number, for you won't know where it is stored. This addressing is done automatically in languages such as FORTRAN and FOCAL.

b. We will select the address 0001_8. Push up the right most switch (key). Push the Load Address button. This tells the computer that you wish to load the number into address 0001_8. ($000\ 000\ 000\ 001_2$). The right most indicator will light. Note: NEVER use the address 0001 on the actual computer.

c. Now set the switches to 1025_8. ($001\ 000\ 010\ 101_2$). This is illustrated in fig. 1. You must make the conversion to binary, for the computer does not accept octal. Push the Deposit button. Whatever was in address 0001 is erased and the number 1025 is stored in 0001.

d. Suppose you continue to load other numbers into other addresses. You then decide to look at what is stored in 0001. (Of course, if you or another student has not used 0001, your original number will still be there.) As in b. above, you Load Address 0001, push the Examine button. Your number (1025) will then appear in the lights. The switch register will not change, however, and will still be set at the last number you set it for (0001).

e. There are 4096 addresses available, from 0000 to 7777. However, on the actual computer you may not use 0000, 0001, and 0010 to 0017. These addresses can hold numbers ranging from -2048 to 2047 (base 10). (-3777 to +3777, base 8, with bit 0 indicating the sign), OR 0000 TO 7777 IF UNSIGNED.

f. The DEC Instruction Manuals number the bits from left to right, starting with bit 0. This is not the same as the powers of 2 in binary arithmetic, which start on the right with 0.

g. Exercises

1. Convert the following octal nos. into binary so that you could enter them into the computer.

ADDRESS	CONTENT	ADDRESS (BINARY)	CONTENT (BINARY)
0050	0002		
0051	0004		
0100	7200		
0101	1050		
0102	1051		
0103	7001		
0104	7402		

2. Make an octal and binary table like that above to store the decimal nos. 16, 18, 20, and -4 into 4 successive addresses starting at address 0026_8.

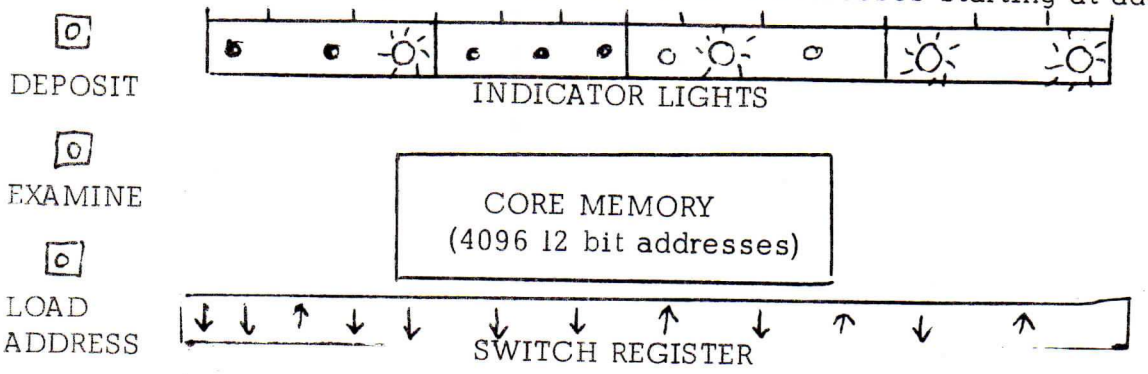


FIG. 1

4. Console operation

a. Console controls

1. Switch register - as explained in Sec. 3

2. Instead of the push buttons mentioned in Sec. 3, we have switches. In addition to those explained in Sec. 3 we have START, CONTINUE; and STOP which are self explanatory.

3. Always be sure that SING STEP and SING INST are depressed. Their function will be explained later. The 6 switches on the left of the switch register must always be depressed. They will never be used on our computer.

b. Console indicators

1. Program Counter - shows current addresses being used by the program.

2. Memory Address - shows addresses being held by the computer during memory cycles. At present it is of no use to us.

3. Memory Buffer - shows the last word which moved into or out of memory. This is where you look after using the examine key.

4. Accumulator - shows current contents of the accumulator.

5. Link - the extra bit on the left of the accumulator provides a 13th bit to be used by the accumulator in certain calculations. It is not considered part of the accumulator. If you are told to check to see if the contents of the accumulator are zero, you disregard the link.

6. The other single light indicators (such as IOT, CLA, etc.) will be explained later.

5. Administrative Procedures for Operations Center.

1. One student will be given permission to use the operations center during a period. This programmer is responsible for enforcing these rules and for allocation of hardware.

2. The programmer in charge may allow upx one other programmer into the center. No visitors allowed, except with permission of a teacher. Those who do not cooperate with the programmer in charge should be reported to Mr. Holzwarth or Mr. Rosen.

3. No waste paper will be left lying around the center. There is a trash container. USE IT, or be barred from the center.

4. No programmer may use the center without permission of on of the following teachers:

Mr. Rosen	Mr. Holzwarth
Mrs. Murphy	Mr. Katz

5. All tapes and manuals must be drawn in advance from Mr. Holzwarth. Whoever draws them is responsible for returning them. If lost or damaged, you will pay for them.

6. Do not feed low speed (yellow) tape into high speed reader.

7. Do not flip the tape reader on and off so as to avoid overflow in FOCAL.

8. Do not disturb the computer when another is operating it.

9. Do not make duplicates of MASTER tapes without Mr. Holzwarth's or Mr. Katz' permission. This is a waste of tape.

10. Do not disturb nearby classes. Remember that the center is not insulated against noise.

11. Be sure that teeth are engaged in advance holes. If a reinforcing patch has been applied, start the tape on the patch.

12. Remember that the total hardware cost $17,000 and that it costs approximately $25.00 per hour plus parts and mileage charges to service it. Tapes cost $2.00 to replace. Manuals cost $1.00 - $2.00 to replace. Although service is under contract and therefore free, a service call causes problems. When a computer is down, others are deprived of its use. Too many service calls may force DEC to revise upwards the cost of the next year's service policy. Service is not instantaneous. It may take up to a week for a difficulty to be diagnosed and corrected.

13. Do not deliver the keys to anyone except Mr. Holzwarth or the next individual who has signed up to use the operations center. Failure to follow this rule usually results in the computer being out of operation for one to four periods while the keys are tracked down.

14. Do not copy or execute a program from the FOCAL Manual. They have been debugged by experts from DEC. Any errors are typographical. There is no instructional value in your playing dice with the computer. If you want to play games, buy a deck of cards and learn solitaire.

6.1. Matrices and 2 dimensional subscripted variables

A. A matrix is a 2 dimensional array of numbers. The numbers are arranged in rows (horizontal) and columns (vertical). An mXn matrix is one that has m rows and n columns.

B. It is necessary to use an artificial system to represent a matrix in FOCAL, since FOCAL can only accept 1 dimensional subscripts. If the computer is powerful enough it would be possible to transcribe a matrix directly by using double subscripts, where the first subscript represents the row and the second the column. An algorithm for representing 2 dimensional subscripts in 1 dimension is clearly explained in the FOCAL Manual on p. 4-9, 10. A system for 3 and 4 dimensional arrays is also included, but we will not use this.

C. If you are entering a simple 2 dimensional table, such as a tax schedule, it might be possible to rearrange the table into 1 dimension before entering it and use it that wya. In most cases, it will be necessary to use the algorithm, e

D. The limits of a subscripted variable are ± 2047. This does not give us 4094 possible nos. in the array. Appendix C of the FOCAL Manual gives a method for estimating the number of locations available for an array. It also gives a method for finding that number after entering the indirect program. Note that if you want the spaces (not locations) open f r a table of numbers. you should not multiply I by 5 in the TYPE statement.

E. Exercises

Given the following matrix A:

1	5	7	6	8
2	3	4	10	9
12	15	14	11	19
20	22	24	23	21

1. Using the method on p. 4-9,10, identify 7 in 2 dimensional notation. In 1 dimensional notation.

2. Do the same for 11.

3. What numbers are located in the following locations: (2 dimensional notation)
   A (0,3) A (3,4)
   A (3,1) A (1,4)

4. What numbers are located at the following locations (1 dimensional notation)
   A(8) A(16)
   A(11) A(19)

5. What is I, IMAX, and J for 24?

6. What is I, IMAX, and J for 3?

7. FOCAL illustration
Write a program to compute and print L and S in the following formulae:

$$R = \frac{b}{a}$$

$$L = ar^{n-1}$$

$$S = \frac{lr - a}{r - 1}$$

Note: For explanation of R, L, and S see any Alg. II text, A=1st term, B=2nd term and N=number of terms.

8. Debugging a FOCAL Program

Most computer programs require one or more debugging runs. Minor errors, such as those due to mistakes in typing and mistakes in format, will be detected by assemblers and compkers. Most logical mistakes cannot be detected by assemblers and compilers. Most logical mistakes cannot be detected. Your first indication of a mistake is when the wrong answer, or no answer (infinite loop) appears. When this happens you must debug the program. The PDP8/S has a DEC Debugging Tape (DDT) for this purpose to be used with PAL III and MACRO-8. The debugging feature of FOCAL is included in the FOCAL tape. For FORTRAN you must devise your own debugging procedure and include it in the program. It can be removed after debugging by deleting those lines.

a. The general rules for debugging any program include:

1. A run to test all possible branch points. Data which will take every branch should be entered. Data that will lead to values that are<1 should be entered, as well as simple integers. Where possible you should know the answers in advance. If everything goes OK you are done. If not, procede to the following steps.

2. TRace. Starting from the beginning the execution of every command is indicated in a print out. In this way an infinite loop can be pinpointed. Erroneous branches can be detected. This will tell you where the program went wrong.

3. Examination of registers. If the above step does not help you, you must examine the final content of all registers (and variables) not printed out in the program. Those that are not correct are followed through from the beginning by halting and examining after each change of content. This will usually solve your problem, but if it does not work, you must then halt your program after every step and examine all registers (and variables).

4. If all the above does not pinpoint your mistake, the only suggestion that can be made is to call it a few names. Make sure no ladies are present. You may NOT kick it. This only works for TV sets.

b. Before using the more sophisticated ODT and DDT on a PAL III program, we will try debugging a FOCAL program.

1. The Sorting problem. A common problem in Data Processing is to take a file (a one-dimensional array) and to sort the file in some predetermined order. For instance, suppose we had a file of students filed at random. We might wish to rearrange them by their student numbers, filed in decreasing order (highest number first). It has been said that it is not economic to sort by computer because punch card machines exist that will do this. Nevertheless, sorting is done by computers in the business world quite frequently.

2. On the next page you will find two programs that do not work. Both of them will give a print out without error messages. Therefore, the mistakes are logical, not procedural. Your job is to find the logical mistakes and correct them.

3. The sorting method (there are many possible methods) that the program is supposed to reflect is as follows:

a. The first number in the array of integers is compared with the second. If the first is greater than the second, we procede to the next step. If the second is greater than the first, the larger is placed in the first position and the second position receives the smaller number.

b. After the first number has been compared with the second and an interchange made if necessary, the larger number is now in the first position. We then compare the first number with the third number as in step one and interchange if necessary. This is then continued with every number in the array in succession. When finished the largest number of the array will be in the first position. The rest of the numbers will still be in random order.

c. We then return to step one, except that we start with the second number. After comparing every number below the second with the second and interchanging if necessary, the first two numbers will be sorted properly (if the program is working).

d. The process is continued untill the entire array has been sorted, in descending order from the highest to the lowest. The array is then printed.

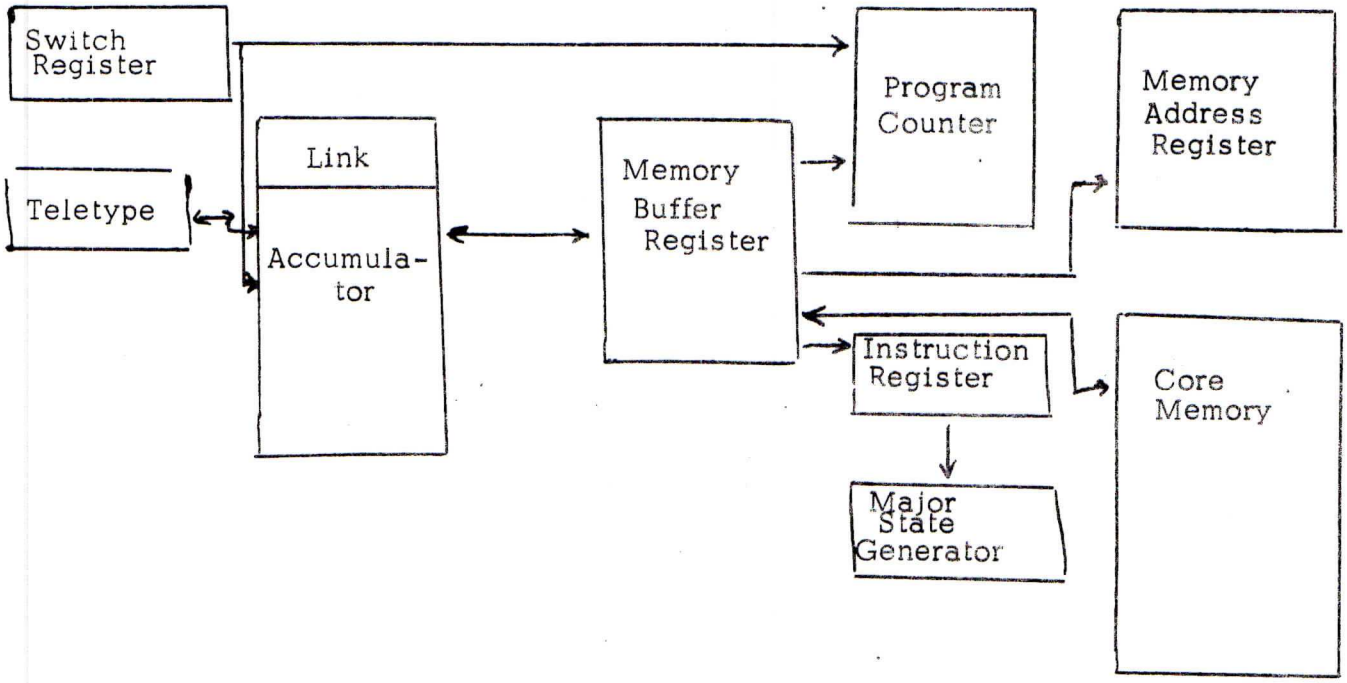
c. For debugging in FOCAL you have the trace feature(see sect. 3.14 in FOCAL Manual). You may also insert TYPE statements at any point. After the program has been halt at a Breakpoint, you may type in direct mode any command, such as TYPE, and thus examine the current value of any variable. Finally you have the GO? command which prints each step as it is executed.

d. Debugging Exercises:

```
1.02 C INCORRECT SORT  ONE MISTAKE
1.03 ASK B
1.04 SET C = B-1
1.05 FOR I=1,B;ASK A(I)
1.06 FOR J=1,B;DO 3.0
1.07 FOR I=1,B;TYPE !,%3.0,A(I)
1.08 QUIT

3.10 SET K=J+1
3.20 FOR I=K,B;DO 4.0
3.30 RETURN

4.10 IF(A(I)-A(J))4.5,4.5,4.2
4.20 SET T=A(J)
4.30 SET A(J)=A(I)
4.40 SET A(I)=T
4.50 RETURN
```

```
1.02 C INCORRECT SORT  TWO MISTAKES
1.03 ASK B
1.04 SET C=B-1
1.05 FOR I=1,B;ASK A(I)
1.06 FOR J=1,B;DO 3.0
1.07 FOR I=1,B;TYPE !, %3.0,A(I)
1.08 QUIT

3.10 SET K=J+1
3.20 FOR I=K,B;DO 4.0
3.30 RETURN

4.10 IF(A(I) -A(J))4.5,4.5,4.2
4.20 SET T=A(J)
4.30 SET A(J) =A(I)
4.40 SET A(J) = T
4.50 RETURN
```

9. Internal organization of PDP8



a. In addition to those explained in 4. above, we have the Instruction Register and the major state generator. They are indicated on the console by the mnemonic coded lights. You will understand them after we have studied the code. As an example, when 1000 was used in the problem in 4. above, the light under TAD was lit. In general, the instruction register holds instructions and the major state generator holds fetch and execution binary codes.

b. The arrows indicate flow of data between the various components.

c. We will now execute a simple program, in machine language. Load Exercise 1 from Sec. 3 into the computer. After entering all of it, load address 0100, and depress START. The accumulator, after a split second, shows 0007, and the computer has stopped.

d. Partial explanation:

OCTAL CODE	RESULT
7200	Clear the accumulator (set = 0.) accumulator.
1xxx	2s complement add the contents of address to the
7001	Increment the accumulator. (Add 0001 to contents of accumulator.)
7402	Halt

e. Exercises:

1. Can you explain what happened? What arithmetic was done? What numbers were combined and how?

2. Assume the following program has been entered into memory.

ADDRESS	CONTENT
0100	0003
0200	7200
0201	1100
0202	3300
0203	7402

Note: 3xxx means deposit contents of accumulator in address xxx and then clear the accumulator. Trace the course of the addresses and contents through the components of the computer. Disregard the instruction register and major state generator. Start from the point when you load the address 0200 into the program counter.

10. Memory Reference Instructions

a. In the examples used in sections 3. and 4. above, we used binary numbers (in octal code) for instructions. It is easier to remember what instructions do if we use an alphabetic code based on the function of the instruction. Later you will learn how to enter this symbolic code into the computer. For the present, we will use the symbolic code for programming and textual explanation, but will translate it into binary before entering the program into the computer.

b. Memory reference instructions are so-called because they operate on the core memory. They require an operand (the memory address to be operated on).

c. Instruction List

MNEMONIC CODE	OCTAL CODE	OPERATION
AND	0000	logical AND
TAD	1000	2s complement addition
ISZ	2000	increment and skip if zero
DCA	3000	deposit and clear AC
JMS	4000	jump to subroutine
JMP	5000	jump

d. The operand is appended as a 3 digit octal number to the first digit listed above. For example, 1350 would result in the content of address 0350 being added to the content of the AC and the sum stored in the AC. The content of 0350 would not change. Symbolically we would write TAD 0350.

e. The operations ARE not fully explained above. Following is a detailed explanation of each operation:

1. AND Uses the following truth table on the binary content of the address and the AC and stores the result in the AC

A ∧ B	= C
0 0	0
1 0	0
0 1	0
1 1	1

2. TAD sec. d. above

3. ISZ First the content of the address is incremented by one and the result stored in the address. Second the next instruction is skipped if the new contents of the address are 0000. If the new content is not 0000, the next instruction is not skipped.

4. DCA The content of the accumulator is stored in the address and the AC is cleared.

5. JMS The content of the PC is stored in the address. The next instruction is taken from the next address. For instance, if PC = 1764 and the instruction is 4104, 1764 would be stored in 0104 and 0105 would be entered into the PC. The next instruction to be executed would be the instruction stored in 0105.

6. JMP The address is entered into the PC and the next instruction is taken from the address.

f. Exercises:

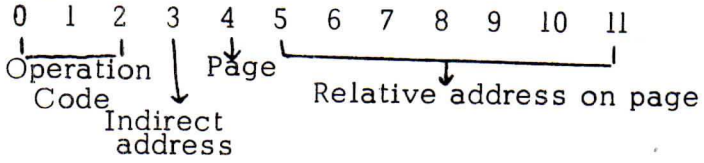
1. Code the following instructions into octal, then into binary.
   TAD 260
   ISZ 220
   AND 321
   MMP 175
   DCA 377

2. Decode the following into symbolic language
   101 001 000 000
   011 001 110 111
   001 000 111 100
   100 000 101 110
   010 001 000 011

II. Paging and indirect addressing

a. MEMORY ADDRESS FORMAT

● BITS



The diagram indicates the functions of the various bits of an instruction. As you read this sheet, be careful to keep in mind whether the text is talking about an instruction or an address. The PDP8 uses a convention that eases the problem of addressing (although you may not think it is easier). The 4096 addresses in the memory are divided into 32_{10} (40_8) pages, each of which contains 128_{10}(200_8) locations, numbered 0 to 177 (octal). Each absolute address has a corresponding relative address. The following table will be helpful.

PAGE	ABSOLUTE	RELATIVE	PAGE	ABSOLUTE	RELATIVE
0	0 - 177	0 - 177	20	4000-4177	0 - 177
1	200-377	0 - 177	26	5400-5577	0 - 177
2	400-577	0 - 177	36	7400-7577	0 - 177
3	600-777	0 - 177	37	7600-7777	0 - 177
4	10000-1177	0 - 177	0	10	10
5	1200-1377	0 - 177	3	617	17
6	1400-1577	0 - 177	3	717	117
7	1600-1777	0 - 177	12	2463	63
10	2000-2177	0 - 177	12	2577	177
11	2200-2377	0 - 177	31	6255	55
12	2400-2577	0 - 177	37	7777	177

If the table is not at hand, you can determine the page of an absolute address by inspecting bits 0-4 of the absolute address. Bits 0 and 1 give the tens digit and bits 2-4 the units digit of the page number. For example, 5400=101 100 000 000. Bits $0,1 = 10_2 = 2_8$. Bits $2,3,4 = 110_2 = 6_8$. Page is therefore 26_8.

b. Bit 4 is used to indicate to the computer that you wish to use a relative address on page 0 (bit 4=0) or the same page as the instruction (bit 4 = 1). For instance, 5313 = 101 011 001 011 means JMP to location 113 (relative) on the same page. 5113 = 101001001011 indicates JMP to location 113 (absolute) on page 0. Note that 5313 and 5113 are used here as <u>instructions</u>, not address.

c. Suppose you wish to JMP to location 113 on <u>another</u> page, not page 0. There is no way to indicate by bit 4 that you wish to do so. (Try to indicate a JMP from page 5 to page 7 if you're not convinced). In fact, you could not JMP to an address higher than 177, except for a useful convention called indirect address. Disregarding pages for the moment, recall that the first octal digit represents the operation code. This leaves 3 digits for the address. Assuming that you are not on page 13, how would you code TAD 2765 in binary? 12765 is impossible. Indirect addressing to the rescue.

d. Indirect addressing is coded symbolically as I and is indicated in the binary instruction by a 1 in bit 3 (0 if no indirect address). We must use a <u>pointer</u> address which will hold the full 12 bit address of the operand that we wish to operate on. This pointer must be on page 0 or the same page as the original instruction.

1. Example (pointer on page 0)

	LOCATION	SYMBOLIC CONTENT	BINARY CONTENT
	0310	TAD I 0035	001100 011 101
(pointer)	0035	2120	010 001 010 000
	2120	0011	000 000 001 001

The computer would go to 0035, find the address 2120 there, and then add the content of 2120 to the AC. The AC would be increased by 11_8 (9_{10}).

2. Example: (pointer on same page)

	LOCATION	SYMBOLIC CONTENT	BINARY CONTENT
	0310	TAD I 0375	001 111 111 101
(pointer)	0375	2120	
	2120	0011	

3. Example: (Illustrating return from another page)

LOCATION	SYMBOLIC CONTENT	BINARY CONTENT
0310	JMP I 0375	
0311	next instruction after JMP	
0375	2120	
2120	7040	
2121	JMP I 2122	101 101 010 010
2122	0311	

e. When addresses 0010 to 0017 are addressed indirectly, they are incremented by one before their contents are used as addresses. This is useful for large blocks of data.

1. Example:

LOCATION	CONTENT
0010	4777
0500	TAD I 0010
5000	beginning of data block

The computer adds 1 to 4777, giving 5000, then goes to 5000 for the first data word. The next time TAD I 0010 is encountered, the data would be taken from 5001.

f. Remember, unless your program and data can be contained on one page, use page 0 for operands. This will avoid the use of indirect addressing and a return to the main program. In general, you will only need to worry about indirect addressing if you have more than 128 numbers to process.

g. Exercises

1. Write a symbolic and a binary program to put the contents of the AC into address 4123 and clear the AC. Put the original instruction in 4110.

2. Same as 1. except put the original instruction in 2110.

3. Write a symbolic and a binary program to increment CONTENTS OF address 5625 and skip the next address if zero. Put the original instruction in 5610.

4. Same as 3. except put the original instruction in 5610 and the operand in 0150.

5. What does the following program do? What is the answer and where is it stored? START AT 6225

LOCATION	CONTENT
0110	0007
6224	0001
6225	7200
6226	1224
6227	1110
6230	3631
6231	0240
6232	7402
0240	0000

Note: You will find it easier to translate octal code into binary, then into symbolic. 7200 = Clear AC. 7402 = Halt.

6. In Ex. 5 above, suppose the last 3 instructions were:

6231	0240
6232	7402
0240	0000

What would the answer be and where would it be stored?

12. Augmented Instructions

a. This group performs its action on the AC and the Link. They do not require a memory reference. Since they do not require a memory reference, bits 3 to 11 are available for another instruction, subject to certain rules. Strictly speaking, they are all part of the instruction 7000 (symbolically written OPR, but never entered into the computer, even in symbolic code), and are therefore referred to as microinstructions (meaning subinstructions or small instructions).

SYMBOLIC CODE	OCTAL CODE	OPERATION
	GROUP ONE	
NOP	7000	No operation (similar to CONTINUE in FORTRAN)
CLA	7200	Clear AC
CLL	7100	Clear Link
CMA	7040	Complement AC
CML	7020	Complement Link
RAR	7010	Rotate link and AC right one
RAL	7004	Rotate link and AC left one
RTR	7012	Rotate link and AC right two
RTL	7006	Rotate link and AC left two
IAC	7001	Increment AC by one
	GROUP TWO	
SMA	7500	Skip if AC negative
SZA	7440	Skip if AC = 0
SPA	7510	Skip if AC positive
SNA	7450	Skip if AC $\neq$ 0
SNL	7420	Skip if link $\neq$ 0
SZL	7430	Skip if link = 0
SKP	7410	Skip unconditionally
OSR	7404	Apply inclusive OR to switch register and AC, store results in AC
HLT	7402	Halt
CLA	7600	Clear AC

b. Group 1 is designated by a 0 in bit 3. Group 2 is designated by a 1 in bit 3. Therefore Group 1 and 2 microinstructions may never be combined.

c. Micro-programming

1. All the microinstructions consist of a 1 in bits 0,1, and 2, and a 1 in one of the last nine bits. For example, CMA = 7040 = 111 000 100 000. No other microinstruction in Group 1 will have a 1 in bit 6. For example, IAC = 7001 = 111 000 000 001. The two instructions are the same, except for bits 6 and 11. We can therefore combine them by using bits 6 and 11 in the same instruction. We now have

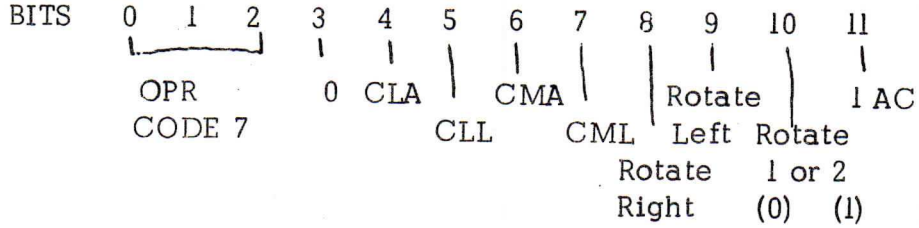
a new instruction CMA IAC = 7041 = 111 000 100 001. The combined instruction has a symbolic code CIA in the symbolic languages, but most combined instructions do not.

2. The sequence of operations is determined by the following event times. It is NOT determined by the order in which you write them.

CLA, CLL	Time 1
CMA, CML	Time 2
IAC	Time 3
RAR, RAL, RTR, RTL	Time 4

If you write CMA CLA it = 111 010 100 000. The AC is cleared first before complementing.

d. Instruction format for Group 1 microinstructions:



e. Exercises:

1. Decode the following combined microinstructions. Convert to binary and use instruction format.

7220
7011
7112
7344

2. Code in octal and binary:

CLA CLL
CMA IAC RTR
RTL CML IAC

13. More on augmented instructions

a. As with Group 1, Group 2 microinstructions may be combined with other Group 2 microinstructions (never with group 1). However, there is a limitation on group 2. Included in group 2 are 2 subgroups - an OR group, which has a 0 in bit 8; and an AND group which has a 1 in bit 8. It is therefore impossible to combine OR and AND group microinstructions.

OR Group	AND Group
SMA	SPA
SZA	SNA
SNL	SZL

Also, SKP cannot combine with either the AND or the OR group(conflict in bit 8)

1. If microinstructions of the OR group are combined in a single instruction, the inclusive OR determines the skip. Truth table for the inclusive OR follows. Values of 0,1 re present the alternatives of No (0) and Yes (1) for the conditions to be satisfied and the output (skip) to occur.

A	V	B	=	G
0		0		0
0		1		1
1		0		1
1		1		1

Example: SZA SNL A skip will occur if either AC = 0 or Link = 1 or both.

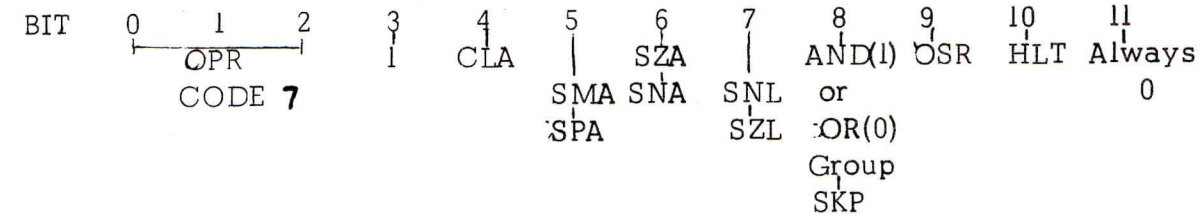
2. If AND group microinstructions are combined, the logical AND (see 10.e.1)) determines the skip. Both conditions must be satisfied, or there is no skip.

Example: SNA SZL. The skip occurs only if AC ≠ 0 and Link = 0

b. Event times for Group 2

All skips	Time 1
CLA	Time 2
OSR	Time 3
HLT	Time 4

c. Instruction format for group 2 microinstructions



d. Exercises:

1. Decode the following microinstructions. Convert to binary and use instruction format

7520
7442
7710
7434

2. Code in octal and binary:

SNA SZL
CLA OSR HLT
SNA CLA HLT
SPA SNA

14 Review of Machine Language

a. Code the following instructions: (into binary, then octal)

*0600

TAD		660
ISZ		620
DCA	I	627
JMP	I	060
AND		721
JMS	I	065
TAD	I	760
DCA	I	773
DCA	I	600
JMP	I	631

b. Decode the following instructions: (into binary then symbolic)

*0600
7000
7702
7636
7460
3460
2170
0721
1234
2345
3456
4567
5677
7373

c. What does the following do?

LOCATION	CONTENT
*300	7300
301	7001
302	2310
303	5302
304	7004
305	7420
306	5302
307	7402
310	0000

d. The following stores the sum of some numbers in a certain address. What is the sum and where is it stored? (All addresses are absolute)

LOCATION	CONTENT
*1000	CLA CLL
1001	TAD 1020
1002	IAC
1003	1450
1004	RAL
1005	SNL
1006	5202
1007	RAR
1010	3603
1011	HLT

(continued)

```
*1020
        4012
        2000
*50
        1021
$
```

e. The following program is executed. What are the final contents of the AC and the Link? (All addresses are absolute.) Original contents of Link=1 AND AC = 1234

LOCATION	CONTENT
*1000	RTL
1001	DCA 1021
1002	ISZ 1017
1003	TAD 1020
1004	NOP
1005	DCA 1011
1006	TAD 1017
1007	DCA 1014
1010	TAD 1021
1011	HLT
1012	IAC
1013	RAR
1014	NOP
1015	RAR
1016	JMP 1015
1017	7401
1020	7006
1021	HLT

$

f. Three numbers are stored in location 300 to 302. Write a symbolic program to subtract each of these numbers from a number stored in 350, and store each of the 3 answers in 351 to 353, respectively. Start your program at 303.

g. Code the following into binary, then octal:

SYMBOLIC	BINARY	OCTAL
2100		
ISZ 2124		
ISZ I 2005		
TAD 2177		
TAL I 2005		
DCA 2027		
DCA I 2150		

h. Decode into symbolic:

OCTAL	BINARY	SYMBOLIC
3450		
0755		
0355		
5555		
5155		
4657		
4257		
4557		

i. Code into binary, then octal:

SYMBOLIC	BINARY	OCTAL
CLA CML RTR		
CLL RAL		
SZA SNL HLT		
CLA IAC		
SMA SZA OSR HLT		

Why isn t it necessary to give a starting address?

j. Decode the following into symbolic:

OCTAL	BINARY	SYMBOLIC
7516		
7320		
7160		
7474		

k. What does the following program do?

```
  200
CLA CLL
TAD 377
DCA 376
DCA I 375
ISZ 375
ISZ 376
JMP 203
HLT
  374
0
500
0
7773
$
```

l. Write a program to enter the contents of 777 into 7 addresses starting at 350 and skipping every other address. Start program at 600

15. The PAL III Assembler

1. The PAL III Assembler translates your source program into binary and punches out a binary object tape. It is similar to the FORTRAN Compiler. It will accept octal numbers as data and as instructions to be stored.

2. Although you should be familiar with programming and flow charts in general, some specific suggestions for PAL III programming include:

a. Use a flow chart before coding.

b. The three general areas of a program follow. You should keep each area separate, in the order indicated.

1. Initialization, After the operating program (2 below) is written, all instructions and registers operated on by the operating program should be set to their initial values. You need not intialize the locations of the operating program itself. The constants and variables (3 below) usually need not be initialized, but the syntax of PAL III usually forces it.

2. Operating program, This includes everything except 1. and 3. It consists exclusively of instruction statements, such as CLA, TAD, etc.

3. Constants and variables, All constants and variables used in the program are assigned locations and listed at the end of the program. Remember that this is not FOCAL, so that you cannot use a constant without first assigning a location for it and storing it. Similarly every variable must have a location assigned.

c. Page 0 is directly addressable from any page. Some candidates for page 0 include:

1. Often used subroutines (such as typying instructions)
2. Constants and variables
3. Subroutine pointers (location pointer should be on the same page if possible).
4. Temporary storage registers.

3. Small sample program

```
        *500
CLA                 / INITIALIZE
TAD 504             / Operating PROGRAM
DCA 505             /OPERATING PROGRAM
HLT                 /OPERATING PROGRAM
1                   /CONSTANT ENTERED
0                   /NOT NECESSARY
$
```

We initialized the AC. We did not initialize 501 or 502 since the operating program was entered there. In the sample shown we need not have initialized 505 since the DCA will destroy its previous contents. On the other hand, 504 had to be set = 1, or the previous content will be added.

4. PAL III rules not previously used for binary instructions

a. All the symbolic codes introduced up to now, such as CLA, DCA etc., may be used.

b. Comments are preceded by a /

c. Names of up to 6 alphanumeric characters may be used as addresses. The assembler will then assign location numbers to correspond to these names. Example: PTR2, 5235. The assembler would store 5235 in a location selected by the assembler and addressable by the name PTR2. DCA I PTR2 would cause the computer to deposit the AC into 5235, using PTR2 as a pointer.

d. * indicates the starting address at which the program will be stored. *500 = start at 500. If the SA is not specified, it will start at 0200. Always put *SA on separate line.

e. $ indicates the end of a program.

f. Auto-indexing - automatic incrementing of addresses 0010 to 0017 only, when operated on indirectly.

5. Exercise:

Problem: Write a flow chart and a program to add up numbers stored in locations 0200 thru 0207 and store the answer in 0410. Start the program at 0600.

16. Using the PAL III Assembler

a. Use the BIN Loader to lead the PAL Assembler
b. Load source tape
c. Turn off punch
d. Turn on reader
e. Load address 0200
f. First Pass
   1. Set SR Bit 0 = 0, Bit 1 = 1
   2. START
   3. If any error printouts, correct source tape and start over. If not, go on to Pass 2.
g. Second Pass
   1. Turn off reader
   2. Load source tape
   3. Turn on reader
   4. Turn on punch
   5. Set SR Bit 0 = 1, Bit 1 = 0
   6. CONTINUE
   7. If error printout, correct tape and start over. If not, go on to (optional) Pass 3.
h. Third Pass (optional)
   1. Turn off reader and punch
   2. Load source tape
   3. Turn on reader
   4. Set SR Bit 0 = 1, Bit 1 = 1

5. Continue
6. Finished

i. Outputs
1. Pass 1 - symbol table on ASR 33 and diagnostics
2. Pass 2 - object tape and diagnostics
3. Pass 3 - Table of locations, and contents in octal and symbolic codes.

j. Execution
1. Load object tape, using BIN Loader
2. Load address SA of program
3. START

k. Exercise

Using the SR, load 1, 2, 3, 4, 5, 6, 7, and 10 (octal) into locations 0200 thru 0207. Using the sample program of 16.2 above, run the program examine 0410 to see if it works.
Note: When Pass 1 is low speed, Pass 2 must be restarted at 02ØØ if high speed object tape is to punched out. (using BIT 11=1)

17. PAL III Diagnostics

a. Pass 1

1. IC ddd AT xxx Illegal Character - It is ignored
dddd = value of illegal character
xxxx = location (not program counter)

2. RD XXXXXX AT dddd Redefinition - This may not be an error.
XXXXXX = symbol being redefined
dddd = location

3. DT XXXXXX AT dddd Duplicate Tag - Symbol is <u>not</u> redefined.
XXXXXX = symbol being redefined
dddd = location
This results from using the same name as an address twice. It only happens when you redefine by the comma. Example: ERROR; CLA
ERROR, TAD 5432

4. ST XXXXXX AT dddd Symbol Table full - revise program
XXXXXX = symbol on which overflow occurred
dddd = location

5. UA XXXXXX AT dddd Undefined address - Assembler makes definition, probably incorrectly for the logic of your program. Revise program
XXXXXX = undefined symbol
dddd = location

b. Pass 2

1. IR dddd AT xxxx Illegal Reference - Assembler corrects references, which may not agree with the logic of your program. Revise program.

dddd = address being referenced.
xxxx = location
This error usually occurs when you attempt a memory reference to a page other than the same page or page 0.

c. Pass 3

1. Illegal Reference - see above

d. Exercises

1. Find the errors and write the error messages that would be produced by the following program:

```
*7300
START, CLA CLI
ISZ CNTR
TAD ADDEN1
TAD ADDEN2
DCA I SUM
START, JMP 300
HLT
TALLY,0
ADDEN1, 1
ADDEN2, 2'
$
```

18. PAL III Print Out Subroutine

a. The following subroutine can be used to print out the contents of any memory location. It will avoid the necessity of loading an address and examining it via the SR. It is not essential that you understand the IOT instructions such as TLS. These will be taken up later.

b.

```
_________
.........
_________
.........
```

(your program which stores the answer in a known location)

```
_________
_________
_________
/THE NEXT 5 INSTRUCTIONS MUST BE INCLUDED IN THE MAIN PROGRAM
TLS → CLA CLL
      TAD xxxx (xxxx is the known location to be printed out)
      JMS 140
      HLT (or continue the main program)
```

```
/THE SUBROUTINE FOLLOWS
*140
ISODET, Ø

RAL
CONT; TAD TEMP
RAL
RTL
DCA TEMP
TAD TEMP
AND MASK
TAD DECODE
TSF
JMP .-1
TLS
CLA
ISZ CNTR
JMP CONT
TAD ZERO
DCA TEMP
TAD M4
DCA CNTR
JMP I ISODET
ZERO, Ø
M4, -4
MASK, 7
DECODE, 26Ø
CNTR, -4
TEMP, Ø
```

c. Exercise: See if you can improve on the above subroutine by shortening it. You must test your subroutine before submitting it. You might find it easiest to test it by starting you program at the CLA CLL and substituting LAS for the TAD XXXX.

19. Symbolic Tape Editor (low speed)

a. The Symbolic Tape Editor is used to correct and/or generate a symbolic language tape in ASCII Code. You will find it best to create your tape off line, but do not correct any mistakes. Then enter your tape into the memory via the Editor and make your corrections on line. The Editor minimizes stray characters and makes corrections easier, both mis-spelling and mis-typing errors and logical errors.

b. The Editor is loaded by the BIN Loader. The SA is 0200

c. Loading an existing tape into the buffer.
1. Put tape to be corrected into ASR 33 reader
2. START on console
3. Type R followed by RETURN Key. If ? is typed, repeat this step.
   a. RETURN key will hereafter be abbreviated as RET:
   b. No line feed is necessary after a RET used after a command
4. Turn on reader.
5. If tape has no CTRL/FORM punched at its end, enter this by the keyboard. Depress CTRL key, then hit FORM key. Bell should ring. Hereafter, CTRL/FORM WILL BE abgreviated as C/F.

d. Punching a corrected tape (already entered into buffer). Note that leader and trailer are not automatically generated. Sequence starts after 5. above.
1. Punch OFF
2. CONTINUE at console
3. Type F, RET
4. Punch ON
5. CONTINUE. Stops on END indicator light. Leader is punc hed.
6. Punch OFF
7. CONTINUE
8. Type P, RET
9. Punch ON
10. CONTINUE. Program is punched out.
11. Punch OFF
12. Type F, RET
13. Punch ON
14. CONTINUE. Trailer id punched.

e. The Editor has 2 modes: Command and Text. In command, all input is interpreted as a command. In text, ■■ text is be entered into the buffer (and applied to your program): To enter text mode, R or one of the editing commands (see below). To return to command mode, you must use the C/F. A common mistake is to enter commands in the text mode. In this case they are entered into the buffer and become part of your program, which usually results in an error message from the compiler or assembler (not the Editor). The bell signals that you have re-entered command mode. A non-existent or incorrect command is answered by a 7.

f. Editing a tape. After entering your tape into the buffer, you may wish to correct mistakes, delete, and/or add lines. The following commands will accomplish this. Most commands to the Editor take the following forms:

X, RET - Perform Operation X
nX, RET - on line n
m, nX, RET - on lines m through n inclusive

The line numbers are not typed by the Editor, nor by you. The Editor appends them in the buffer only. You must count to see which line you are referencing. **The line numbers are numbered decimally, starting with one.**

1. nD or m,nD - Delete line or lines. Editor remains in command mode.

2. nI - Insert text before line n. You may enter as many lines as you wish. Editor remains in text mode. To return to command mode, use C/F.

3. nC or m,nC - Change line or lines. Original lines are deleted by the editor. Type in the correct lines in full. Editor remains in text mode until you enter C/F.

4. K - Kill the entire buffer. Punching a tape does not remove it from the buffer. To enter a new tape and edit it, use the K command.

5. L , nL , or m,nL - Prints the entire buffer (L) on ASR 33, or a selected line (nL), or lines (m,nL). After editing you might check to see if your edited program is correct.

g. There are many other commands and short cuts available with the Editor. They will be covered later (maybe). Meanwhile you are referred to the manual.

h. Exercise:

```
*650
CLA CLL
JMS I 250
DAC Y
TAD TEST
CIA
TAD 668
DAC RSLT
HALT
$
```

Find the errors in the program above. Write commands that you would use to correct them with the SYMBOLIC TAPE EDITOR.

20. Symbolic Tape Editor (High Speed)

a. The High Speed Editor is loaded by the BIN Loader. The SA is 0200

b. Put tape to be corrected into ASR 33 Reader; if oiled.

c. Put tape to be corrected in to high speed reader, IF UNOILED.

d. After loading 0200 SET THE OPTIONS:

BIT	POSITION	ACTION
10	0	Low speed output.
	1	High speed Output
11	0	Low speed input. Tape in ASR 33 reader
	1	High speed input. Tape in high speed reader.

e. START on console

f. Type R, RET

g. If tape has no CTRL/Form punched on it, enter this via ASR 33

h. Correct tape. See sheet 19

i. Turn on high speed punch

j. Type T, RET

k. Type P, RET

l. Type F, RET

m. Type T, RET

n. Notes:

1. The command T, RET does not give a trailer code, nor does the F, RET. Blank tape is punched out. This is accepted as HIGH SPEED INPUT by the FORTRAN Compiler and the PAL III Assembler, and FOCAL

2. Whenever the console lights indicate END in the above sequence, such as after P, RET, use CONTINUE at the console to procede with the sequence.

o. Exercise:

Enter the incorrect tape of the exercise in Sheet 19. Correct it and punch out a corrected symbolic tape.

21. Review - PAL III

a. Ten numbers are stored in locations 200 through 211. Another number is stored in 212. Write a program starting at 600 that will determine and print out how many of the ten numbers are larger than the number in 212. Use a flow chart.

b. In the program below, what is the output of Pass 1 ?

```
  650
BEGN, CLA CLL
TAD 600
CIA
DCA COUNT
LOOP, TAD 601
ISZ COUNT
JMP LOOP
HLT
COUNT, 0
$
```

What is the octal code for JMP LOOP ?

What are the octal contents of BEGN ? of LOOP ?

22. Systems analysis

a. The following job classifications are used in industry for data processing personnel. They may vary within the industry, generally in omission of subclassifications. Each job is subclassified in large installtions into the following levels: (listed in descending order of reponsibility and pay)

1. Manager or supervisor- in charge of all activities of section or job.
2. Assistant manager or supervisor
3. Lead,- has full technical knowledge of section and supervisory duties over the following subclassifications.
4. Senior - works at highest level of all technical phases of activity, usually working on his own. May give some guidance and direction to lower levels.
5. Junior - Fairly competent to work on several phases of activities with only general directions, but still needs some instruction and guidance for other phases.

b. Job classifications (which may be subclassified by above, such as Lead Programmer, Senior Programmer, etc. Listed in descending order)

1. Manager of all data processing activities

Plans, organizes, and controls all activities listed below. Personally handles major personnel, administrative, and EDP problems.

2. Systems analyst

Establishes and implements new or revised systems and procedures of EDP. Prepares feasibility studies and systems designs. Formulates logical statements of business problems and devises procedures for solutions. Analyzes existing system difficulties and revises logic and procedures as necessary. Develops logic and procedures to provide more efficient machine operations. Prepares system flow charts to describe existing and proposed operations.

3. Programmer

Establishes standards for block diagramming, machine flow charting, and programming procedures. Writes and debugs programs. Works closely with systems analysts. Verifies program logic by preparing test data for trial runs. Prepares instruction sheets for guidance of computer operations.Translates flow charts into coded machine instructions. May assist in determining causes of computer malfunctions. Evaluates and modifies programs to allow for changes in system requirements or equipment configuration. Prepares documentation of all procedures and programs.

4. Operator

Monitors and controls computer by operating central console. Collects and loads data files, records, programs needed for runs. Controls switching in and out of auxiliary equipment. Maintains equipment logs and other records pertaining to computer operation. Maintains records and supervises inventory of stores and supplies needed in operations center. Follows documentation and instruction sheets under general supervision of programmers.

c. The weekly salary ranges for the above as of 1969 were:

1. Manager - 267 to 373. Phila. avg. 301
2. Systems analyst - 163 to 340. Phila. avg 235
3. Programmer - 131 to 308. Phila. avg. 200
4. Operator - 104 to 240. Phila. avg. 155

d. Educational requirements increase as classification does. Although only high school graduation is required for entry as junior programmer, further classification advancement requires at least 2 years post-high school in most firms. Most systems analysts hold a degree in math or scienceor engineering, plus post-BS (BA) study.

e. The following simplified outline (to which many exceptions could be taken) will help clarify the different job classifications:

1. The manager or identifies a large problem and asigns personnel to work on it. He lays down the guidance and restrictions within which they work.

2. The systems analysts design documents and prepare overall flow charts for flow of information through the system. They also prepare flow charts for the computer programs, assisted by the programmers.

3. The programmers take the computer flow charts, revise them if necessary (in collaboration with the analysts), and construct and code the programs. They debug the programs and run test runs. They then prepare documentation and instructions for the operators.

4. The operators run the programs.

f. Many times, especially in small firms, the dividing lines between the various classifications are non-existent or blurred. I know of one firm in which the entire EDP staff consists of a combined manager, analyst, and programmer, assisted by one operator.

g. You will be divided into groups and given a simple systems analysis to do. You may not think so, but it will be easy. It will also be time consuming. Tjis will give all of you a better understanding of how EDP in industry works. The following outline will aid you in planning your work.

1. Preparation for the study of the system.

a. We will omit most of this. Among other items, this phase should define the problem(s) to be analyzed. The problem that we will analyze is:

See Committee assignments

I . In industry, the problem might be to convert a payroll to Automatic Data Processing, or an inventory control, or an oil refinery scheduling, etc.

b. You must find out what management wants and needs to know. This must be kept in mind through all subsequent phases of the analysis. In industry, too much output is almost as bad as too little. You must interview management. For this analysis management will be Mr. Rosen and the teacher in change of the exam you are analyzing. I will help you with any mathematical or statistical formulas that you need. Just write down what management wants and them come to me. For example, you will probably be asked for the standard deviation, which none of you have heard of.

2. Study of the present system. In industry this would require studying and visiting many subdivisions of the company or department and interviewing all key personnel.

a. Collection of documents now in use.
b. Interviews to determine handling, processing, forwarding and use of the documents.
c. Flow chart of the present system. You should have two-one to show the flow of documents and another to illustrate the computer program.

3. Design of the new system

a. Determination of output. You will be guided in this by 1.b, above.
b. Determination of input requirements. What do you need to know to get output? As an obvious example you will need to know how many students get each raw score.
c. Master file content. This is for industry only, for we will not need to store the information for future use. If this is requested in your interview with management, it is my decision that we cannot do it, for our computer configuration is not designed for this. For your own information, we could do this by punching the data onto tape and saving the tapes.
d. Design of records and reports. See a. and b. above. You must design the format of all records - input and output - including the form(s) on which the teachers will report the results.
e. Flow chart for the new system. In industry you would also prepare a flow chart for the physical handling, processing, etc.

of records. You need only prepare the computer program flow chart for this analysis.

f. Construction of the computer program. You may use FOCAL; FORTRAN; PAL III; or MACRO - 8.

4. Implementation of the analysis

a. Documentation by programmer. Industry only. This is obviously at our level.

b. Controls. Industry only. This promotes accuracy in input data and in the operation of the system. We could use this, but it would increase the scope of the problem too much. An example at an elementary level is the Checksum in our tapes, which detects errors in the tape input.

c. Files conversion. Industry only.

d. Organization and staffing. Although you might wish to, you may not fire any of the teachers, including the DP instructor. You are also stuck with the committee assigned.

e. Evaluation. Debug your program by entering test data and making any necessary corrections.

f. Maintenance. Industry only. How do we make changes to keep the system up to date?

5. All the steps listed above should be assigned completion dates. The DP instructor and you will work together on this. In industry this area alone is the subject of many books.

d. The systems analyst must possess a questioning and analytic mind, use an objective approach, and be patient. In industry he must be able to sell his ideas. The cost to industry of originating and assembling data for a computer has been estimated at three times the cost of the computer. The potential for savings by designing a better system is tremendous. You are about to undertake one of the most challenging and rewarding tasks in the data processing field. Good Luck.

23. PAL III Type in Subroutine

a. The attached subroutine will allow you to enter any octal number into any address. The subroutine should be approached in the main program as follows:

_____your program_____

```
HLT      / Signals you to start typing. First hit CONTINUE:
JMS 10
```

_____rest of your program_____

b. Using the attached subroutine you should enter the address first, followed by a /, followed by the desired contents. NO SPACES.

Example - 2ØØØ/1234 Will store 1234 (octal) in register 2ØØØ.

c. Exercises

1. See if you can shorten the subroutine
2. Write a program, using the type in and type out subroutines, that will allow you to enter a number, multiply it by 4, and print the answer. See Sect 18 for print out subroutine.

24. Revi ew PAL III

A. The following program is supposed to multiply the 2 nos. entered at NUM1 and NUM2. Nos. to be entered via SR. Why won t it work ?

```
  200
CLA CLL
JMS 20
JMS 20
TAD NUM1
CIA
DCA CNTR
LOOP, TAD NUM2
ISZ NUM1
JMP LOOP
DCA PROD
JMS 140
TAD PROD
HLT
CNTR, 0
NUM1, 0
NUM2, 0
PROD, 0
$
```

B. Write a program to enter 2 positive octal numbers, divide the smaller into the larger, and print the answer and remainder. Assume that the larger number is always entered into a known location and that the smaller is always entered into another known location. It will not be necessary to have the program select the larger of the two numbers.

25. Using ODT to debug a PAL III or MACRO-8 program

A. We have studied debugging FOCAL programs. Many fo the principles are the same, but the detailed instructions are different . Until now, if a PAL III program did not work, you could use the SR to examine the contents of registers. You could use the Single Instruction Switch to halt the program and procede step by step, monitoring the temporary contents of locations. This is a slow process. The greatest delay occurs when you have to create and assemble new source tapes. If your corrected tape does not produce the desired results, you must assemble another corrected tape until either you are successful or jump out the window.

B. ODT allows you to enter programs, test them, enter and test corrections, and punch out a correct object program without reloading any tapes. Its only drawback is that it communicates in octal code, not symbolic. Another program (DDT) which we will study later communicates in symbolic language, but it requires more operations to load and more memory space. Furthermore, we have had difficulties in getting the breakpoint (see below) to operate and this is a very important debugging tool.

C. Preparations for debugging

1. Collect the binary (object) tape of your program, the output of Pass3 of PAL III for your program (which gives the octal contents of all addresses), a and the ODT tape.

2. Load ODT with the BIN Loader

3. Load your object program with the BIN Loader. It must not overlap ODT. ODT occupies locations 1000 to 1577 (Low) or 7000 to 7577 (High).

4. Load address 1000(Low) or 7000 (High).

5. START at console.

D. Most useful commands:

1. reg/ This causes the register (octal) to be opened and the contents to be typed in octal code. A new code may now be typed and it will be entered into the location.

2. Return This key closes the register. If no new code has been entered the previous code will be retained. If you do not close the register, your next commands will destroy the contents of the register.

3. Line Feed This closes the register and opens the next sequential register.

4. regG This transfers control to user s program at the register specified. Usually used to start execution of your program, in which case use the SA of your program as the specified register. After you have transferred control to your program, you must restart ODT.

5. regB This halts your program at the register before the one typed. You must start your program before the breakpoint will take effect. Use regG. When the breakpoint is reached, the program halts before executing the contents of the register specified. The register number is then typed, followed by a (, then the contents of the AC. You may now open, examine, and modify other registers. You may insert another breakpoint, but only ONE breakpoint can be in effect at any time.

6. C Typing C after a breakpoint halt has taken effect will continue execution from the breakpoint. NOTE: If your program contains input-output, which we will study later, the program will hang up in an infinite loop. A patch must be used. See Instruction Manual for ODT for the patch.

7. B This removes all breakpoints.

8. T Punches leader code Follow procedure below exactly.
   a. Type T
   b. Punch ON
   c. STOP at console
   d. Punch OFF
   e. If going on to 9. below, restart ODT at 1000

9. location1;location 2P Punches binary program on ASR33 only, between limits of location 1 and location 2, inclusive.
   a. Type location 1 (4 octal digits); location 2P. Computer halts.
   b. Punch ON
   c. CONTINUE
   d. Punch OFF before any further commands, including 10. below.

10. E Checksum and trailer
   a. Type E. Computer halts
   b. Punch ON
   c. STOP at console
   e. Punch OFF

E. Exercise

Debug the following program. It is supposed to square a number located in 0600 and place the answer in SQ. It does not work. Using ODT, debug it. You will find it easiest to open 0600 and place your inital values there by using ODT.

```
        650
BEGN, CLA CLL
TAD 600
DCA COUNT
LOOP, TAD 600
I       ISZ COUNT
JMP LOOP
DCA SQ
HLT
COUNT, 0
SQ, 0
$
```

26. Average Computer

A. The next concurrent assignment will be to construct a PAL III (or MACRO-8) program to print out the cumulative average for each student at the end of each report period.

B. Input-output

1. After each test, the mark and the total of checks and axes , since the previous test, are input. Previous cumulative marks, number of tests, and totals of checks and axes are also input.,
2. After each update, current totals are output.
3. On signal, cumulative mark is output.
4.

B. Computations

1. Cumulative test marks are added and divided by no. of marks.
2. Two checks add one, two axes subtract one from test means.
3.

D. First requirement

1. Program the above in FOCAL, using only 4 students. Other variables may be any values > 1.
2. Omit temporarily B.2. above.

27. Input--Output Lowspeed

A. We have a low speed input-output device (ASR 33) and a high speed (750). They share many I/O instructions and the basic principles of flags. Each of the 4 devices has micro-instructions usable only for that device. There are no special instructions for the low speed punch. The type instructions are used and the punch turned on.

B. Flag principles

1. If the flag (an electro-mechanica; component of the I/O devices themselves) is set (up) the device may be used. If it is an input device, it will enter the info into the computer. If output, it will do its thing. (type or punch). If the flag is cleared (down) the device will not operate. Set = 1 and clear = 0.

2. In general, the programmer must set (raise) the flag in his routine. When the computer is turned on, the flag may be set or cleared. Therefore most routines using I/O start by **initializing the flag or flags, using TLS and KCC.**

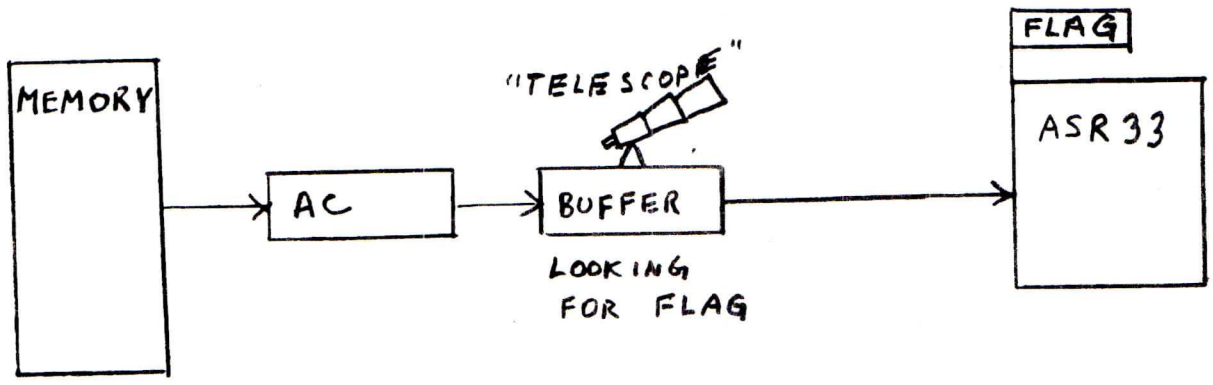
3. After using a device the flag will automatically be set.

4. Each device has a skip instruction. It should be followed by a JMP.-1 instruction. If the flag is set, the JMP.-1 will be skipped and the routine will procede. If the flag is cleared, the routine will return and check the flag again. This will continue until the flag is set. Since the devices are much slower than the computer, this may take many loops while the device is handling only one character.

5. Usually the necessary clearing of the flag while operating a device is done automatically. Most I/O instructions are combined and one of the microinstructions clears the flag. For instance KRB (see below) not only accepts input from the ASR 33 keyboard, but it clears the AC first and clears the flag. After the character has been read in, the flag is automatically set.

6. With more experience, you might be able to program the computer to do something while waiting for the flag. At present, you had better have the computer wait.

C. To illustrate, we will assume that a character stored in memory is to be printed. The following are involved:



1. The character is TADed into the AC. Under appropriate instruction the character (represented by a binary 12 bit number) is loaded into the output buffer. HOWEVER, if the flag is not set, the instructions TSF and JMP .-1 will cause it to continue jumping back and forth until the flag is set.

2. Why wouldn t the flag be set?

a. You forgot the TLS in the initialization section of your program. When you came on line, the flag was down and it will remain down. Result is an infinite loop between the TSF and the JMP .-1.

b. The ASR33 is not finished typing the previous character. The buffer is not clear and the flag stays down until the buffer is cleared.

3. Finally the flag is set and your character procedes through the buffer to the ASR33. Electro-mechanical circuits type the single character.

4. Why not omit the TSF and JMP .-1 ? Because the spped of the computer would cause it to go through many instructions (some of which might be I/O) while the ASR33 is typing one character. In this way much I/O could be lost.

5. NOTE: The binary characters are not outputted, for they are the equivalents of the stray characters to the ASR33. You see them being printed by Pass 2 of PAL III (low speed). Sections 28 and 29 will explain how to get octal equivalents of the binary contents printed.

D. Remember that the teletype will not type unless you tell it to. It is possible to "type" characters into the computer without having them appear on the teletype print out sheet. It is not recommended.

E. ASR 33 I/O instructions. Read in from keyboard or reader. Flag is set by key board, indicating key has been pressed.

SYMBOL	OCTAL	RESULT
KSF	6031	Skip if flag - 1
KCC	6032	Clear AC and flag (=0)
KRS	6034	Read keyboard/reader buffer
KRB	6036	Clear AC, clear flag, read buffer

F. ASR 33 I/O instructions. Print out on teletype sheet (and punch if desired). Flag is set by keyboard, indicating it is ready to print.

SYMBOL	OCTAL	RESULT
TSF	6041	Skip if flag = 1
TCF	6042	Clear flag
TPC	6044	Load buffer, (from AC), select and print.
TLS	6046	Load buffer (from AC), select and print, clear flag. (Reset flag when finished)

G. Exercise

Write a routine to type out the 4 octal codes located in locations 600 to 603. Assume those codes have already been entered. Start your routine at 650.

23. ASC II Codes

A: All the standard keyboard characters are handled by the computer as code. When you type A, for instance, this goes into the keyboard buffer as 301 (octal) and into the computer as 000 011 000 001. It is stored in this manner. Whenever the printer buffer contains 301, under control of a type out subroutine, the teletype prints the A.

B: Following is most of the ASCII code:

CHARACTER	CODE	CHARACTER	CODE	
A	301	0	260	
B	302	1	261	
C	303	2	262	
D	304	3	263	
E	305	4	264	
F	306	5	265	
G	307	6	266	
H	310	7	267	
I	311	8	270	
J	312	9	271	
K	313	(	250	
L	314	)	251	
M	315	/	257	
N	316	=	275	
O	317	LF	212	(Line feed)
P	320	CR	215	(Carriage Return)
Q	321	Space	240	
R	322	+	253	
S	323	-	255	
T	324			
U	325			
V	326			
W	327			
X	330			
Y	331			
Z	332			

C. Exercises

1. What 4 codes would be necessary for the instruction 7420?
2. What codes would be necessary to have the computer print out the word HELLO with a line feed and carriage return after printing it?
3. Develop a code for the Roman Numerals I, II, III, and IIII (= IV)

29. Input--output of alphanumeric characters

A. As noted in 27. (above), input characters are automatically converted to an 8 bit code on the device buffer. These 8 bits may then be inclusively ORed into bits 4 to 11 of the AC under control of a KRS instruction. Note that bits 0 to 3 are not disturbed. Should you have any content in the AC, the resultant number will not be the same as the code in the device buffer. For instance, if AC = 7001 and the character A (code 301) is read in under a KRS, you will end with 7362 in the AC. To avoid this, use KRB or use CLA before a KRS. If you enter 2 or more characters, each character is entered and processed separately.

B. The greatest difficulty is with output. Bits 0-3 are lost when the AC is dumped into the device output buffer. Should you be attempting to print a 12 bit number (not an 8 bit code), you will lose bits 0-3 and bits 4-11 will be read as ASC II code. For instance, suppose you have the number 7563 stored in the AC (111 101 110 011). Under control of a TLS, you will not get 7563 printed out. It will be truncated to 01 110 011 (163) in the buffer, and this corresponds to no code, so nothing will be printed. As another example, 1363 stored in the AC would be printed under control of a TLS as the character C. It is impossible to get more than one character printed out at a time without programming for this.

C. The solution is to isolate each octal digit (3 binary bits) from the 4 octal digits and print it separately. We will first consider the isolation method.

a. The first octal digit must be represented by bits 0-3.

b. If we rotate to the left 4 times (don't forget the Link) we will then have the first octal digit stored in bits 9-10-11. The second octal digit will then occupy the Link and bits 0-1.

c. Applying a mask of 000 000 000 111 under control of the inclusive AND, we will then convert all but the last 3 bits to 0. We could then print this, although not without further processing.

D. Exercise

Write a routine to isolate and "print" all 4 digits in turn. At the point in the routine where you would print it, just write "print".

30. Input-output of alphanumeric characters (cont.)

A. In 29 we saw what instructions were necessary to select each digit of a 4 octal digit number and to enter each digit into the AC. Remember that 0003 in the AC is not printed under control of a TLS as 3. The code for 3 is 263, therefore 263 must be entered into the buffer.

B. The 8 octal ditits (0-7) that are possible correspond to the codes 260-267. If we add 260 (000 010 110 000) to the digit in the AC we have converted it into the necessary code. For instance, 0 + 260 = 260; 1 +n260 = 261; etc. This constant (260) must be added by your program.

C. In a similar manner, a number x enters the AC under control of a KRB as 26x. Subtraction of 260 will convert the code to the number that is to processed in the computer.

D. In ut of a complete 4 octal digit number is the reverse of what has been discussed for output. After decoding, the first digit is rotated 3 places to the left and then entered into a assembly location. The second digit is then entered into the AC and the number in the assembly location is TADed. After rotation, the result is then returned to the assembly location and the process is repeated until the complete 4 digit number has been assembled. It is then available for use.

E. Exercises

1. Return to the exercise of 29. Where you wrote print insert the proper instructions to decode and printout the number.

2. Write a routine to enter a 4 octal digit number from the keyboard into location 700. Start your routine at 650.

31. Review of Input -- Output

A. Thus far we have considered how to enter a number from the ASR 33, and how to have this number printed out. We will review input-output by a problem that will call for both these actions and will perform 2 of the elementary arithmetic operations. When you have finished it you will see why the FORTRAN and FOCAL tapes are so long.

B. Exercise

1. Write a program to enter 2 numbers via the ASR 33, add or subtract the numbers (depending on whether they are separated by + or -), pring an = sign, and then print the answer. Follw the answer by a CR/LF.

2. Limitations and directions

a. Let both numbers be 4 digit positive octal integers < 3500.

b. Let the subtrahend be $\leq$ minuend. (Avoids negative answers).

c. Start your program wherever you wish. However, if you plan to use DDT or ODT, you must avoid certain locations. As usual, do not use the locations occupied by the RIM or BIN Loaders.

d. Do not include error messages to take care of mistyping.

33. Introduction to MACRO-8

A. Macro-8 is a language that is an extension of PAL III. It is titled Macro-8 because it includes macro-instructions (literally large instructions) that include in a single instruction the equivalent of 2 or more PAL III instructions. Many of these macro-instructions are expressed by a single ASC II character. All the PAL III instructions are included in the Macro-8 language.

B. The basic ideas of flow charting and programming that we studied heretofore are still applicable. The first 4 chapters of the Macro-8 manual duplicate the material that we have already studied in PAL III. We will confine ourselves to learning the definitions of new Macro-8 instructions and using them in programs and routines. These programs and routines will be similar to those that we have already done in PAL III. The new material will consist of using the new instructions to shorten our programs and routines. With one important exception I/O is exactly the same in Macro-8 and PAL III.

C. As an illustrative example, let us start by a simple exercise and use a few new Macro-8 instructions to program it. The following characters must be defined:

CHARACTER	USE
+	Addition of symbols or integers
-	Subtraction of symbols or integers
;	End of coding line. Allows use of more than one instruction on the same line. Does NOT combine the instructions. May not be used to terminate a comment.
=	Defines parameters.

D. Two more features must be explained:

1. Integers may be used directly. Although they must be stored in a register, this will be done by the Macro-8 assembler and need not be done by the user's program. When so used, they must be enclosed by parentheses. For instance TAD(5).

2. Constants may be used directly. Although they must be stored in an address, this will be done by the M-8 assembler. When so used, they are enclosed in parentheses. Example: TAD (5)

3. Constants need not be assigned locations. Unlike PAL III, VARIABLES must be assigned.

E. Exercise

Write a program to count how many of the 5 numbers stored in locations 400 to 404, incl.,are greater than 7. This count should then be deposited in TALY.

34. Macro-8 Operating Instructions and Diagnostics (Low Speed)

A. Macro-8 is a 3 pass assembler like PAL III. It comes in high and low speed versions, both of which are exclusively high or low. You cannot use the ASR 33 with the high speed version as we can with PAL III. We will study low speed first.

1. Pass 1 outputs diagnostics only
2. Pass 2 outputs the binary object tape, followed by the symbol table. If you want the symbol table for use with DDT, leave the punch on after the binary tape trailer appears. Otherwise, turn the punch off after trailer code appears.
3. Pass 3 is optional and outputs the register number, the octal contents of that register, and the symbolic code for the contents for every step in the program.

B. Operating instructions

1. Load Macro-9 tape, using BIN Loader.
2. Load address 0200
3. Place source in reader and turn on reader
4. Set bit 0 = 1 for pass 1
5. START
6. After Pass 1 finishes, turn off reader.
7.
8. Reload source tape and turn on reader.
9. Turn on punch
10. Set bit 1 = 1 for pass 2
11. CONTINUE
12. After finishing pass 2, turn off punch and reader
13. Reload source and turn on reader. Load Address 0200.
14. Set bit THREE (not 2) = 1 for pass 3
15. START

C. Diagnostics. Format is CODE, ADDRESS. The address will be given as absolute octal or relative to the last symbolic tag before the error.

CODE	INTERPRETATION
PE	Too many instructions and/or literals (constants) on page.
ZE	Same as above, but refers to page 0.
ID	Illegal redefinition of a symbol, not using =. For instance A,5 and A,7 occurring in the same program. However, it is possible to write A=7 after A,5 located at same address other than 7. The reverse (= before ,) is an error.
IC	Illegal character was detected and ignored.
IE	Illegal use of equal sign, such as A+B = C
II	An out of page reference was made in an indirect address
LG	An out of page reference was made in a direct address
SE	Symbol table exceeded. Your symbol table overlaps the internal symbol table of Macro-8.
IM	Illegal format in macro definition
US	Undefined symbol, such as TAD C and C not defined.
MP	Missing parameter in Macro call. too many literals.
BE	Two Macro-8 internal symbol tables have overlapped. Usually ^

D. Exercise

1. Assemble the exercise in 33 above.

2. Expand the exercise to include I/O. Restrict I/O to positive octal nos. $\leq$ 2047

35. Macro-8 Character list and tabulations

A. In addition to the letters and digits already familiar to you from PAL III, Macro-8 includes the following characters:

CHAR	USE
Space	Combines symbols or numbers in certain context, otherwise no effect. For instance, CLA CLL. This ORs the two instructions
+	Combines symbols or numbers - addition
-	Combines symbols or numbers - subtract
!	Combines symbols or numbers - OR
=	Defines parameters
;	Terminates coding line
&	Combines symbols or numbers - AND
"	Generates ASCII constant
()	Defines literal on current page
[ ]	Defines literal on page Ø
<>	Defines a macro instruction - user defined

B. The following characters are ignored, but do not produce diagnostic IC

CHAR	USE
Form-feed	Generated by Symbolic Editor
Blank tape	High Speed leader - trailer
Code 2ØØ	Low Speed leader-trailer
Rubouts	Erasure
Line feed	Format for source tape

C. All other characters are illegal and will produce the diagnostic IC (except in a comment or TEXT field).

D. Tabulations. The only use of tabs is to provide a neat format for the program listing. A tab is not equal to a fixed number of spaces, but is set for a fixed location on each line. The following rules are generally used.

1. A title comment, or comment standing alone, begins at the left margin
2. Pseudo-instructions are usually indented 1 tab stop. The first tab stop is 10 characters from the left margin
3. Address tabs begin at the left margin, followed by a tab before the instruction field.
4. Instruction fields are indented 1 tab stop
5. A comment is separated from the preceding field by a tab, unless space

(10 char from 1st Tab Stop.)

is lacking. The second tab is approximately 3 inches from the left margin. The tab does NOT replace the symbol /.

E. Exercise

1. Using the symbolic tape editor, produce a high speed source tape from your low speed solution to 33 above. (see 33.1). Include the proper tabs and additional coding to have the program reinitialize itself, i.e. reset counter, tally, and any necessary instructions.

36. Pseudo-instructions

A. Pseudo-instructions are directions to the assembler to perform certain tasks or to interpret subsequent coding in a certain way.

B. Location counter

PAGE n will reset the location counter to the first address of page n. It facilitates the segmenting of long programs.

PAGE with no argument will reset the location counter to the first address of the next page, i.e. if it occurs at 457, the location counter would be reset to 600.

C. Radix control

DECIMAL This allows you to switch from octal to decimal base numbers. All numbers after the appearance of DECIMAL will be taken as decimal until the occurrence of the pseudo-instruction OCTAL.

OCTAL This returns you to the octal base.

D. Double precision integers (CONSTANTS ONLY)

DUBL This allows you to increase the range of the numbers used from (-2048 to +2047) to (-8,388,608 to +8,388,607). The numbers are stored in 2 consecutive locations. Unfortunately they may not be combined with operators as literals. For instance, it would be incorrect to use TAD (4587). It would also be incorrect to write TAD A if we stored the number at A because it would occupy the location A and the location following A. Double precision numbers are always taken as decimal base, but the radix for the rest of the program is not changed. After appearance of DUBL, all numbers are taken as double precision integers until the appearance of an alphabetic character. Example:

```
          *400
DUBL      679467
          44
END,      CLA CLL
```

The assembler would produce the following:

LOCATION	CONTENTS	
400	0245	
401	7053	(since $679467_{10} = 2,457,053_8$)
402	0000	
403	0054	(since $44_{10} = 54_8$)
404	7300	(Appearance of the E in the END, has cancelled the DUBL)

E. Text

1. A single character preceded by a " is treated as its 8 bit ASCII code. For instance, TAD ("A) would enter 301 into the AC.

2. TEXT A string of text characters may be entered by giving the instruction TEXT, followed by a space, a delimiting character, the string of text, and the same delimiting character. The delimiting character may be any alphanumeric character. / is suggested. The text characters are trimmed to 6 bits and stored 2 to an address. Following the last character a 6 bit zero is stored as stop code. You must be careful in outputting the characters, for you must separate the 6 bit characters and add back the first digit (3).

Examples:

```
   *400   TEXT/NUTS/
results in
0400    1625
0401    2423
0402    0000
```

```
* 600
TEXT    TJOET
results in
0600    1217
0601    0500
```

F. Floating point constants

FLTG This instruction allows use of double precision floating point constants in E format. As above, they may not be used with operators and are taken as decimals (current radix of program is unchanged however). They occupy 3 registers and are stored in normalized form (see 38. below). All numbers after FLTG are taken as floating point constants until appearance of an alphabetic constant (other than E). This increases the range of usable constants to -999.999E99 to +999.999E99

Example:

```
    *400
FLTG     509.32E 03 would produce the following coding:
0400    0003
0401    2427
0402    6670
```

2. Tne details of the coding will be studied later. For now you need only realize that the part of the number following E (called the exponent) is stored in the first register and the other 2 registers contain the rest of the number (called the mantissa).

3. The "floating point" refers to the decimal point. In all numbers up to now it was fixed to the right of the rightmost digit. With this pseudo-instruction you may place your decimal point anywhere within the limits of the format.

G. Exercises

1. Write a routine to add 2 positive double precision integers. In your coding indicate the registers to be used for the numbers and the answer. Do not include any input or output coding. Do not program for possible overflow of most significant bits into bit 0. Do program for overflow of least significant bits into link.

2. Write a program to add positive floating point numbers. Indicate registers used for the numbers and the answer. Do not consider the overflow into bit 0. Assume that both numbers have same exponent. Omit input and output.

37. Floating point package

A. All the floating point and decimal pseudo-instructions covered up to now allow you to program constants, but do not allow you to program decimal variables, particularly input-output variables. There is a set of 4 programs and tapes provided by DEC that permit you to use decimal base numbers for any purpose. The methods used to convert decimal base numbers to binary must be understood before you can use the packs intelligently.

B. Floating point standardization example.

1. We will work an example in decimal base first since it should be easier for you to follow after your instruction in scientific notation.

2. Before arithmetic can be performed on 2 nos. they must be converted to a standard normalized form. This consists of lining up the decimal points and making the exponents equal. Let us add 50 and 0.75. The first step is to convert all integers or mixed integer-decimals to pure decimal. We have:

$$\begin{array}{rl} & .50 \times 10^2 \\ + & .75 \times 10^0 \end{array}$$

We now rewrite the numbers by converting the powers of 10 so that they will be equal, but still preserve the pure decimal form:

$$\begin{array}{rl} & .50 \times 10^2 \\ + & .0075 \times 10^2 \end{array}$$

Addition now gives the result of $.5075 \times 10^2$, or 50.75

Had there been any zeros in the answer between the decimal point and the first non-zero digit, the number would have been normalized by removing them and adjusting the exponent.

C. Conversion of decimal number to normalized binary.

1. Convert the no. to fixed point binary

$$25_{10} = 11001_2$$

2. Convert the fixed point binary number to normalized floating by moving the decimal point to left of the left most digit. Count number of places moved. This is the exponent. (of 2)

$11001_2 = .11001 \times 2^5$ Point moved 5 places, so exponent is 5.

Check:

$$\begin{aligned} .11001 \times 2^5 &= (1 \times 2^{-1} + 1 \times 2^{-2} + 0 \times 2^{-3} + 0 \times 2^{-4} + 1 \times 2^{-5}) \times 2^5 \\ &= (1/2 + 1/4 + 1/32) \times 2^5 \\ &= 25/32 \times 2^5 \\ &= 25 \end{aligned}$$

3. Store the exponent in the first address of a 3 address block reserved for the FP number. Store the mantissa in the second address, starting at bit 1. Bit 0 is reserved for the sign. If more than 11 bits are needed, continue storing in the 3rd address, after the 2nd addres is full.

Loc 1	000 000 000 101	$= 0005_8$
Loc 2	011 001 000 000	$= 3100_8$
Loc 3	000 000 000 000	$= 0000_8$

D. Conversion of negative numbers is the same, and will be shown here as a further example. Convert -.09375 to normalized binary:

$$\begin{aligned} 0.09375 &= -.00011_2 \\ &= -.11 \times 2^{-3} \end{aligned}$$

Exponent = -3 = 7775_8

Mantissa = 111000000000 = 7000_8
Stored as:

7775
7000
0000

E. Exercise

Convert the following decimal nos. to floating point normalized binary:

10

35

2049 (=4001)

-9

0.75

F. Commands in the Floating Point Package #1

1. Like all machine language programs, the FP Packs work with an AC, where all arithmetic takes place. However, the regular AC only handles 12 bits, therefore a floating AC (FA) is set up at locations 44, 45, 46. There is no floating link, nor do the various skips, rotations, etc. operate on the FA.
The following commands are the on ly ones we will use:

FGET A Loads contents of A, A+1, A+2 into FA.
FPUT A Transfers contents of FA into A, A+1, A+2. Does not clear FA
FADD A Adds conten ts of A, A+1, A+2 to contents of FA
FSUB A Subtracts from
FMPY A Multiplies by
FDIV A Divides contents of FA by contents of A, A+1, A+2
FEXT Exits from FP Interpretative Mode (see below)

G. FP Interpreter

1. Before using any of the above commands, it is necessary to enter the FP Interpretive Mode. This is done by JMSing to a subroutine starting at 5600. Before using any further PAL III or MACRO-8 instructions, you must exit from the Interpretive Mode by the commands FEXT. Once you have entered the Interpretive Mode, you need not reenter for <u>consecutive</u> FP commands.

H. FP I/O

1. Data may be entered by using a subroutine starting at 7400. Data may be typed in the format $\pm$xxx.xxxE$\pm$xx . Any of the signs, the decimal point, and the entire exponent may be omitted. The data goes into the FA.

2. Data will be outputted in the above format. You must first enter a subroutine starting at 7200. The contents of the FA will be typed.

I. There is much more that can be dome with the FP Packs. If you wish to use them, you must borrow the manual and the tapes.

J. Exercise

Program the entry of 2 decimal nos., addition of the nos., and output of the decimal answer.

38. User defined Macros

A. In a program certain sequences may be used several times with just the arguments changed. In these cases the sequence can be defined with a macro name and generated thereafter by that name, plus arguments. It is similar to a subroutine, except that subroutine arguments cannot change.

B. A macro may be defined without arguments. In that case, itis practically the same as a subroutine.

C. The macro is defined by use of the pseudo-instruction DEFINE, followed by the Macro name and a list of dummy arguments. The sequence of coding then follows enclosed by ◇ ,

Example:

```
*500
DEFINE MOVE DUM1 DUM2                          (Use of spaces, not commas, is critical)
<CLA
TAD DUM1
DCA DUM2
TAD DUM2>
```

A calling sequence for the above macro would look like this:

```
*400
A, 0
B, -6
MOVE A,B    (Note use of commas here)
```

This would result in the following coding:

```
0400    0000
0401    7772
0402    7200
0403    1200   (beginning of macro)
0404    3201
0405    1201
```

In the above example the real arguments A (400) and B (401) have been substituted for the dummy arguments.

D. Restrictions

1. Macros may not be nested, i.e. another macro may not appear within a macro, nor may a macro be used as an argument in the calling statement.

2. TEXT or " may not be used within the macro, nor used as real arguments.

3. The symbols used as dummy arguments may not be used elsewhere in the program.

4. A macro may not be redefined. Use of an argument other than the dummy arguments is a redefinition. For instance, in the above example, we could not inswert DCA X, for X was not included in our list of dummy arguments. It is irrelevant whether X was previously defined in the main program.

5. The calling statement must include exactly as many real arguments as there are dummy arguments.

E. Exercise

Write a program that will add 4 successive pairs of numbers tored in locations 400 to 407 and store the answers in locations 500 to 503. Let the numbers be positive single precision integers. Ignore possible overflows. Use a macro for the addition.

39 Review of Macro-8

Exercise: Write a program that will accept a dividend, a /, and a divisor from the keyboard, peform the division, and print the quotient preceded by an equals sign. Let the numbers be single precision positive octal integers. Let the remainder if any, be truncated. No round off.

The following is the output of Pass 3 for the solution to the above program. Note that the program uses a user-defined macro, a text output, and literals. These are features that are not available with PAL III.

40. Introduction to Statistics

A. Statisticians are concerned with data that have been obtained from taking observations, in the form of measurements or counts, from a source of such observations. For example, in studying the quality of autos, a number of autos would be selected and tested for quality; or, in studying public opinion a small percentage of the population would be selected and questioned. The purpose is to draw conclusions about the source of the observations. For instance, the small number of autos are tested to determine the quality control of the entire output of the company. Public opinion is sampled to determine the opinion of the entire population on a controversial issue. The set of observations that is taken from some source is called the sample. The source is called a population. Statistical methods may be described as methods for drawing conclusions about populations by means of samples.

B. Samples should be taken by random sampling. This is a separate topic that could be studied in detail. It will be sufficient to state here that it is a method whereby individual pieces of data are selected haphazardly, so that no individual member of the population has a better chance of being selected than any other. We should not study the weights of students in our school by collecting our data from the football program, for this would exclude all non-team members, and also would bias our results toward the heavy side. We should not collect the data from DP class, for this would exclude all junior high students (and some groups of senior high as well). For ease of gathering data we may use the latter, but the results will be statisitically invalid for the school.

C. The weights of 100 students are listed here:

Was this sample large enough to represent our 5000+ students? This question should also be studied, but we will ignore it.

With so many items in our sample we are lost in details. Grouping the data into classes will make it easier to study. The boundaries of each class should be selected to give 10 to 20 classes. By subtracting the lowest measurement from the highest, we get the range. Dividing the range by 10, we get the class interval. The midpoint for each interval is defined as the class mark. Should we find that the class interval is an inconvenient figure, such as a decimal in the above example, we should adjust it accordingly. No harm will be done as long as the intervals are kept equal.

D. We now construct a frequency table of the data. The frequency (f) is defined as the number of measurements in each class. The sum of the frequencies should equal the total number of measurements (n).

CLASS BOUNDARIES	FREQUENCY	CLASS MARKS

E. A bar graph, called a histogram, may now be constructed. Use the class boundaries as the horizontal axis and the frequencies as the vertical axis.

F. Exercises

1. Finish D. and E. above
2. Draw a histogram for the following frequency table:

BOUNDARIES	f
155-157	4
158-160	8
161-163	26
etc.	53
	89
	146
	188
	181
	125
	92
	60
	22
	4
	1
	1

41. Mean and Median

A. If we wish to compare different sets of data we could make individual frequency tables or histograms and then compare them. This might be difficult, especially where the differences are not large, or the general shape of the histograms are not similar. For this reason various arithmetical descriptions have been defined. These allow us to compute a number for each set of data and then compare the numbers. Probably the best known is the mean, commonly called the average by non-mathematicians.

B. The mean is defined as

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{k} x_i f_i$$

where there are k classes and x_i, f_i, and n are defined as in the preceding section.

1. There are methods available for shortening a hand claculation of $\bar{x}$. These are of little value for a computer calculation. Our problem is to program the calculation. It is easiest to add all the individual measurements, keeping an internal count of n and then divide by n. If n is known, no internal count need be kept. Thus, for unclassified data the formula becomes:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

2. If a frequency table is desired, an array should be set up and a tally entered for each score or class. The calculation of $\bar{x}$ could procede simultaneously.

C. The median is defined as the middle measurement after the data have been arranged in order of magnitude. If there are n values, the median is the (n+1)/2 the value, counting from top or bottom. If n is even, we use a point halfway between the n/2 and (n+2)/2 scores.

1. The calculation of the median is easy if the data are previously sorted. However, the time consumed by a sort in FOCAL will prevent you from calculating the median for unsorted data.

D. Exercises

1. Program the calcualtion of $\bar{x}$ and the median for classified data which have already been arranged into a frequency table. Assume n is unknown.

2. Program the construction and printout of a frequency table from unsorted data. This would be too time consuming in FOCAL, except that we will also assume 10 classes with a class interval of 10 and a range from 1 to 100.

42. Standard Deviation

A. There are many uses for the standard deviation (s). Only one will be considered here. It is a measure of the variability or scatter, of the data. In other words, how close the measurements are to each other and how closely they are grouped about the mean. For example, consider the sets of data 4, 5, 6, 7, 8 and 2, 4, 6, 8, 10. Both have a mean of 6. However, the second set is more widely spread from each other and from the mean. Calculation of s for both (see below) yields 1.08 and 3.16, respectively. As another example, if 2 classes took the same test and the mean for both was 75, but the s for one was twice the other, one would suspect that the class with the lower s displayed a better knowedge of the material tested. In fact, a very large s would lead one to suspect that the test did not really measure what it was supposed to. The reason for the word suspect is that there are many other factors which might have lead to the aforementioned results.

B. The standard deviation is defined as:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{k}(x_i - \bar{x})^2 f_i}$$

For unclassified data, replace k by n and delete f_i.

C. Although it is possible to program the above formula; a much easier one to use is:

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{x}^2} \text{ (unless) or } \sqrt{\frac{1}{n}\sum_{i=1}^{k} f_i x_i^2 - \bar{x}^2}$$

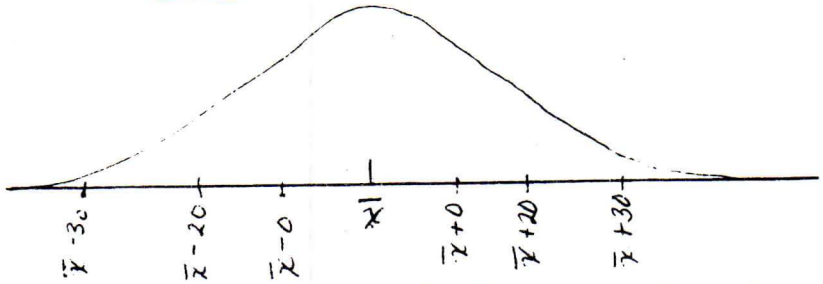
D. Exercises

1. Add to the program for the mean (preceding section) the calcuation of s.

2. Add to the program for the frequency table (preceding section) a print out of the histogram. Have the program type an asterisk for the frequency in each class. Print the class mark and then the asterisk next to them.

43. Normal distribution

A. If we make certain assumptions that are not discussed here, but which are crucial, it can be stated that a histogram of tables of data will approach the normal distribution graphed below. It is also called the binomial distribution. One of the assumptions is a <u>large</u> amount of data.



B. Some populations that follow this distribution are heights of students, IQ's of students, measurements of industrial products, and ages at which people die. Strictly speaking, certain statistical tests must be met.

C. The areas under the curve, or the percentage of the total population is distributed as follows:

1. 68% between $\bar{x}$ - s and $\bar{x}$ + s
2. 95% between $\bar{x}$ - 2s and $\bar{x}$ + 2s
3. 99.7% between $\bar{x}$ - 3s and $\bar{x}$ + 3s.

D. If we construct a histogram of test marks, even using class intervals, it would not approach the above ideal curve. There are many reasons for this. Can you suggest any? If we arbitrarily apply the above percentages to the marks, the result <u>will</u> follow the normal distribution. This is what students call "curving" a test. For ease of calculation, the following percentages are used:

E. Exercise

Given the test marks to be announced in class, redistribute them into a normal distribution, using $\bar{x}$ and s. Then redistribute into the percentages in D. above.

44. Sampling and Estimatimation

A. This topic will end with one of the many possible applications. For other applications and further study you are referred to Hoel, Elementary Statistics, in the Math Department Library.

B. Consider the following problem:

A manufacturer of autos knows from experience that the compressability of auto springs is approximately normally distributed, and that the standard deviation remains fairly constant at 20 lbs. He wishes to estimate the weight that will completely compress the spring (bottom the suspension) from a new batch of springs. He tests a random sample of only 25 springs and finds the mean weight that will compress the spring is 3000 lbs. How accurate is the sample estimate for the population of all the springs? Note the similarity of this problem to the current Chevrolet slogan of computer selected springs.

C. New symbols must be introduced. We use μ for the population mean and σ for the population standard deviation. We retain $\bar{x}$ and s for the sample mean and standard deviation, respectively.

D. Many different samples could be drawn from a population, each with its own $\bar{x}$ and s. It can be proven that the mean of individual means is also normally distributed and that the formula for the standard deviation of the sample means is:

$$\sigma_{\bar{x}} = \frac{\sigma}{\sqrt{n}} = \frac{20}{\sqrt{25}} = 4$$

E. From the normal distribution curve of the previous section we see that approximately 95% of the area of the curve lies within 2 standard deviations of the mean. In other words, 95% of the sample means ($\bar{x}$) will lie within 2 standard deviations of the population means. Therefore the manufacturer can feel 95% certaub that the population mean will lie within 8 or less lbs of the sample mean. The population mean (u) will lie within 2992 and 3008 lbs. If we then apply σ to this μ, we find that the manufacturer can feel 95% of 95% or 90.25% certain that his springs can be rated for a maximum load of 2952 to 3048 lbs.

F. Suppose the manufacturer is not satisfied with 90.25% certainty, but desires 97%. The table in C gives the area under the normal curve for various values of the standard deviation measured in both directions from the mean.

G. Reading the table we see that the closest value we have is .9722 at σ = 2.2. Therefore in the problem we use 2.2 x 4 or 8.8 lbs. The population mean is then 97% certain of being within 2991.2 and 3008.8 lbs. Although not included in the table, 4σ gives almost 100% certaintly. 100% x 97% = 97% certaintly that the springs can be rated within μ + 4σ. Since 4 x 20 = 80, the springs can then be rated within (2991.2-80=) 2911.2 and (3008.8+80 =) 3088.8 lbs.

H. Exercise

These do not lead to good computer problems. Much more accurate tables exist, but their size is beyond the capacity of our computer. The table itself could be computed within the program, but this is beyond your math knowledge, for it involves calculus.

1. Write a program to input a % of certainty, a sample $\bar{x}$, a population σ and the size of the sample. Output is to give the limits of population means for the given % of certainty.

2. Use the program to solve this problem:

Assuming a school population of students is normally distribted with σ = \$1.00. A sample of 100 students is questioned and their mean allowance is \$10.00 per week. With a 90% certainty within what limits will the mean allowance for the entire school lie?

I. Table of Areas Under Standard Normal Distribution Curve

σ	Area
0.0	.0000
0.1	.9796
0.2	.1596
0.3	.2358
0.4	.3108
0.5	.3830
0.6	.4514
0.7	.5160
0.8	.5762
0.9	.6318
1.0	.6826
1.1	.7286
1.2	.7698
1.3	.8064
1.4	.8384
1.5	.8664
1.6	.8904
1.7	.9108
1.8	.9282
1.9	.9426
2.0	.9544
2.1	.9642
2.2	.9722
2.3	.9786
2.4	.9836
2.5	.9876
2.6	.9906
2.7	.9930
2.8	.9948
2.9	.9962
3.0	.9974

45. FOCAL Curve Plotting

A. We have studied the graphing of linear functions. It is possible to have the computer do the plotting. We must consider several questions before writing the program.

1. Is the equation a linear function, i.e. are the exponents 1 ? if not, it cannot be graphed by the methods to follow.

2. Will the axes by placed upon the plot or superimposed later by hand?

3. Related to the above question is scaling. What are the maximum values to be plotted? Your ordinate cannot exceed 72 (or less depending on the location of the horizontal axis). Also there is a problem of distortion due to 5 spaces = 3 linefeed. The latter will be considered later.

B. Method of plotting

1. The equation is written as a function of one of the variables, for instance $4y + 5 = 3x$ is written as $y = \frac{3x - 5}{4}$

2. The approximate locations of the line upon the axis is sketched. Quickly calculating the intercepts will help.

3. Based on the above (and the maximum ordinate of 72) decide on the location of the axis. It is easiest to rotate the graph 90 degrees counter clockwise so that the left edte of the paper becomes the bottom. From this point on we will work under this assumption.

4. Based on the above a tentative decision is made as to the maximum and minimum values to be used for the independent variable (which is then plotted on the horizontal axis). Usually the independent variable is donoted by x.

5. Decide on the increments to be used for the independent variable. Each increment calls for a line feed. Don't waste too much paper.

6. By substitution the values for the dependent variable are calculated at the maximum and minimum values of the independent variable. If they exceed 72 (or less depending on the location of the axes), start over at step 4).

7. Enter the maximum and minimum values and the increment into the statements of the loop used to plot the equation. The basic loop is found in the FOCAL Manual. For any function, I is the independent variable, J the dependent. The right hand side of the equation is used for the upper limit of the dependent variable FOR statement.

C. Illustration, Graph $2y + 1 = 4x$ Numbered steps follow steps in B above.

1. $y = 2x - 1/2 = 2x - .5$

2. Intercepts are (0, -.5) and (.25, 0)

3. Place the axes at the left and bottom of the plot.

4. Try minimum value for x = 1, maximum = 15.

5. At minimum value for x, y = 1.5. This is OK. At maximum x, y = 29.5 This leaves too much waste space at top of plot. Change maximum x value to x = 33.

6. Increment x by 2. This will use 16 lines of paper.

7. 1.1 F X = 1,2,33;T"*",!;F Y=0,2*X-.5;T" "
DO 1,1

D. Exercise

1. Devise changes in the illustration that will place the X and Y axes in the center of the plot. What will you use for the Maximum and minimum values for x?

2. Write the loop statements for the equation 4y + 15 = 8x
3. Write the loop statements for the equation 5x + 3 = 9y

46. Linear Programming - graphic solution

A. Problem: ABC Drug Co. markets cold pills in the following formulas:

SIZE	ASPIRIN	PHENACETIN	CAFFEINE
X	2 gr	5 gr	1 gr
Y	1 gr	8 gr	6 gr

An MD wishes to prescribe at least 12 gr of aspirin, 74 gr of phenacetin, and 28 gr of caffeine. Determine the <u>least</u> number of pills (and what type) he should prescribe.

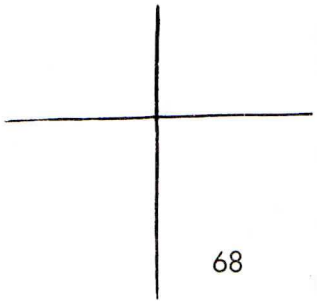
B. Solution

1. Let x = no. of size X pills
Let y = no. of size Y pills

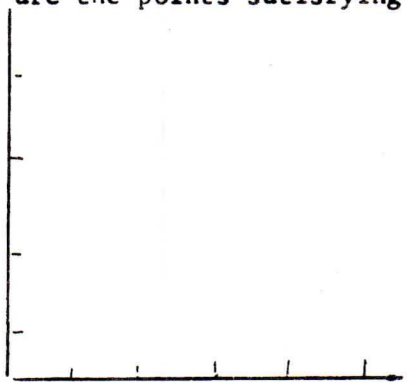
2. What are the constraints (restrictions) on the problem?

(1) $x \geq 0$ (2) $y \geq 0$ (why?)
(3) $2x + 1y \geq 12$ (to get at least 12 gr aspirin)
(4) $5x + 8y \geq 74$ (why?)
(5) $1x + 6y \geq 28$ (why?)
Let m = sum of the no. of each type of pill
(6) $x + y = m$ and m must be a minimum.

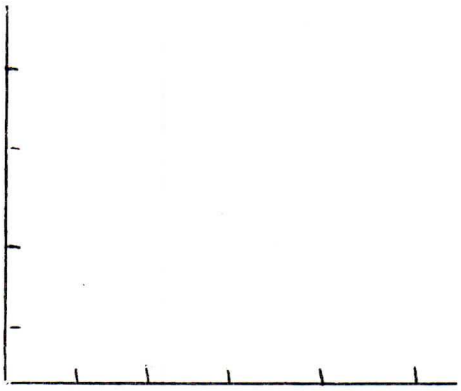
3. Graphically, where are the points satisfying constraints (1) and (2)?



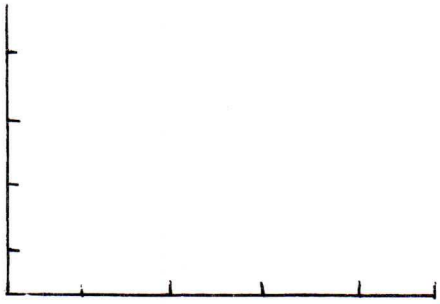
4. Where are the points satisfying constraint (3)



5. Where are the points satisfying constraint (4)

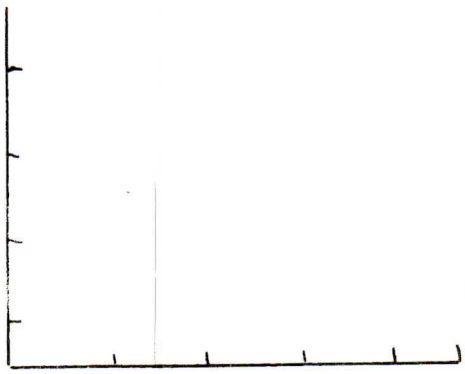


6. Where are the points satisfying constraint (5)



7. All the above represent <u>closed</u> half planes since the boundary points (the line) are included. If the boundary points were not included, they would be <u>open</u> half planes

8. Any solutions must exist within the intersections of all these closed half planes. Plot simultaneously all the lines on a single graph. The answer must lie in the area to the right, called the <u>feasible region</u>.



9. Consider constraint (6). All values for m would result in a series of parallel lines. As m decreases, the lines are displaced toward the left, as m increases, the lines are displaced to the right. Plot various values of m on the axes above, as dotted lines. Include $x + y = 10$.

10. Parts of some of the dotted lines above lie within the feasible region. Any point on these segments would be a <u>feasible solution</u>.

11. In the graph above, $m = 10$ must be the minimum value. $m < 10$ is not in the feasible region. $m < 10$ is not a solution (why?). Therefore we use this equation ($x + y = 10$) and either of the constraints (3) or (4) to form 2 simultaneous equations in 2 unknowsn, which are easily solved. (Why not use constraint (5)). For instance:

$x + y = 10$
$2x + y = 12$, which gives $x = 2$, $y = 8$.

12. Therefore, 2 of size X and 8 of size Y is the minimum no. of pills to be prescribed. Note that this gives an <u>overdose</u> of caffeine (50 gr). This is called <u>slack</u> and is unavoidable. This is <u>optimal solution</u>.

C. Exercise: The lunchroom makes its hamburger and spaghetti in the following formulas:

TYPE	VITAMIN B1	VITAMIN B2
H	1 mg	5 mg
S	4 mg	1 mg

How many helpings of each should you buy to insure an intake of 24 mg of B1 and 25 mg of B2, keeping the food to a minimum.

47. Gauss-Jordan Complete Elimination Procedure

A. In 41 above we saw that it is possible to solve a linear programming problem by graphic methods. It is difficult to get exact answers by this method, especially when non-integral values are considered. Even if this problem were overcome or ignored, the process is time consuming. If we plot by using the computer, accuracy is sacrificed for speed. There is a method which is readily adapted for computers and yet gives very accurate results. It is based on the Gauss-Jordon Complete Elimination Procedure.

B. Problem: Solve the following 3 equations in 5 unknows for u, v, and w in terms of x and y. x and y are then called <u>parameters</u>.

$$\begin{aligned} 2x - y + 2u - v + 3w &= 14 \\ x + 2y + 3u + v \quad\;\; &= 5 \\ x \quad\;\; - 2u \quad\;\; - 2w &= -10 \end{aligned}$$

C. Solution (Gauss-Jordan method)

1. Form a matrix from the coefficients and constant term.

x	y	u	v	w	const
2	-1	②	-1	3	14
1	2	3	1	0	5
1	0	-2	0	-2	-10

2. Eliminate u from each equation except the first.

a. By algebraic manipulation of the entire first row, set the <u>pivet</u> coefficient (circled above) = 1. This is done by dividing each entry in the first row by the pivot.

b. Eliminate the coefficient of u in the second row.

1. Multiply the first row by the additive inverse of the coefficient of u in the second row.

2. Add the 2 rows and use the sum as the <u>new</u> second row.

c. Eliminate the coefficient of u in the third row by the method above, giving a new third row. It can be proven that the new matrix represents a system of equations equivalent to the original system.

d. A repeated process like the above is called an <u>iteration.</u>

e. The matrix now looks like this:

--keep this 1st row to have entry 1 in column 3.

f. Exercise. A new iteration will eliminate v. Starting from the matrix inn e. above, select 5/2 as the pivot.

-4/5	7/5	0	1	-9/5	-32/5	new second row
						new first row
						new third row

Notice that this is not the matrix sine 1st and 2nd rows are interchanged.

g. A new iteration will elimate w. Use -4/5 as pivot

h. Our original system of equations is now seen to be equivalent to the following system, based on the last matrix derived.

$$\begin{aligned} x + y + u &= \\ x + y + v &= \\ x + y + w &= \end{aligned}$$

i. In the above system, express u, v, and w in terms of x and y.

u =
v =
w =

C. Exercises

2. Using the matrix in g. fill in the coefficients and constant terms in h.
3. In h. above express u, v, and w in terms of x and y.

D. In the equations derived on preceding sheet there are infinitely many solutions obtainable by varying the parameters x and y. There is one special solution that we need for linear programming. If we let $x = 0$ and $y = 0$, then $u = 2$, $v = -1$, and $w = 3$. The solution in which the parameters (independent variables) are set = 0 is called a basic solution. u, v, and w are called basic variables and x and y nonbasic variables. We also say that u, v, and w are in the basis.

E. Consider the last matrix formed on the preceding sheet. The constant term (rightmost) column contains the values for the basic variables. We could have read them off without the substitution in D above. We determine which value is assigned to which variable by looking for the entry 1 in the variable column. The value for that variable is located in the same row as the entry 1, in the constant column. For instance, in the v column the 1 occurs in row 2, therefore, we look to the rightmost entry in row 2 and obtain $v = -1$.

F. A column in which all the entries = 0, except one entry which = 1, is called a basic unit column vector.

G. Exercise

Find the basic solution for the following system of equations:

$$\begin{aligned} 2r + 3s + 2u - 2v + 2M &= 240 \\ 2r + s + 8u + 4v &= 120 \\ -5r - 4s + M &= 0 \end{aligned}$$

48. Extended Simplex Tableau

A. Let us suppose that a linear programming problem has led to the following constraints:

$$x \geq 0 \quad y \geq 0 \quad z \geq 0$$

$$\begin{aligned} x + y + z &\leq 100 \\ 3x + 2y + 4z &\leq 210 \\ 3x + 2y &\leq 150 \\ 5x + 4y + 6z &= M \end{aligned}$$

and M must be a maximum.

B. Since the Gauss-Jordan method uses equations, not inequalities, we introduce slack variables u, v, and w and rewrite as follows:

$$x \geq 0 \quad y \geq 0 \quad z \geq 0 \quad u \geq 0 \quad v \geq 0 \quad w \geq 0$$

$$\begin{aligned} x + y + z + u &= 100 \\ 3x + 2y + 4z + v &= 210 \\ 3x + 2y + w &= 150 \\ -5x - 4y - 6z + 0u + 0v + 0w + M &= 0 \end{aligned}$$

This leads to the matrix:

x	y	z	u	v	w	M	const

The matrix represents a basic feasible solution:

x = 0, y = 0, z = 0, u = 100, v = 210, w = 150, and M = 0.
It is not the best (optimal) solution.

C. We will now use an adaptation of the Gauss-Jordon method to find the optimal solution. The method rests on removing one of the variables in the basis and replacing it by one of the nonbasic variables. The column M may be disregarded, for it never changes in the procedure below. The proof of the procedure will not be given. If interested, see Glickman, Theory of Games, in the Math Dept. library.

D. Find the most negative entry in the last row. Its variable will be the new basic variable. We must select a pivot from this column.

1. Divide each entry in this column into the entry in the same row in the rightmost column.

a. In the matrix above -6 is the most negative. Division yields the following quotients:

100/1 = 100 210/4 = 52 1/2 150/0 = ?

b. Select the entry which gave the least quotient as the pivot.
Since 210/4 = 52 1/2 is the least quotient, we select 4 as the pivot.

E. Eliminate z from every equation except the second (as outlined in 43. above). This puts z into the basis and (as will be seen) removes v from the basis.

F. Again find the most negative in the last row and procede as above. This will introduce y into the basis and displace w.

G. At this point there are no more negative values in the last row. This identifies the optimal solution. From it we read off the following values:

x =
y =
z =
u =
v =
w =
M =

H. Exercises

Find the matrix in F. above and fill in the values in G. above.

49. Program for Linear Programming

Write a program in FOCAL that will find the optimum solution for a linear programming set of equations and inequalities.

1. Assume that you have formed the set. The program should start with input of information about the matrix.

2. Have the operator select the pivot. Although it would be possible to have the program do this, it would make the program too long for an exercise.

3. Have the operator select the rows to be transformed.

50. Review of Linear Programming

Solve the following problems, using hand computation or the program in 45.

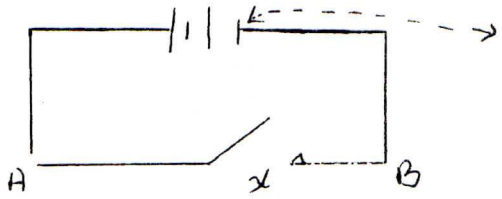
1. The school store sells zip guns at a profit of $.50 ea and switch blades at a profit of $.40 ea. It takes 2 minutes of a salesgirls time and 2 minutes of a cashiers time to sell a zip gun. It requires 3 minutes of a salesgirls time but only 1 minute of a cashiers time to sell a switch blade. The school store operates for a maximum of 2 hours during a school day, and during this time there is 1 cashier and 2 salesgirls available. How many of each item should the store sell each day in order to earn a maximum profit? Z = no. of zip guns B = no. of blades

2. A firm manufactures machine guns and recoilless rifles for student demonstrators. It sells the guns at a profit of $.03 and the rifles at a profit of $.04. Each type is processed on a planer and a miller. The guns require 2 minutes of processing time on the planer and 5 minutes of processing time on the miller. The rifles require 3 minutes on the planer and 2 minutes on the miller. Each machine is available for not more than 60 hours during a week. How many of each type should the firm produce each week in order to make a maximum profit.
G = no. of MG's
R = no. of RR's

51 . Switching Circuits and Boolean Algebra

1. Switching circuits are electrical circuits controlled by switches. As you know, our computer operates in binary. If we include a switch in a circuit and arrange to open or close it, current will flow through the circuit if the switch is closed, but no current will flow if it is open. If we then include a device to measure the presence (or absence) of the current, we can represent a 1 by a measured current pulse and a 0 by an absence of a pulse. We can transport a binary digit along the circuit by carrying it on pulses of current which are sent along the circuit at equal (very small) intervals. It may then be measured or recorded (by indicator lights, for example) and combined with other circuits, all of whose pulses are accurately synchronized. This is a simplified description of the basic building blocks of a computer. In order to understand how a computer is designed, we must study how these switching circuits are combines.

2. We will confine ourselves to simple single pole switches which may be diagrammed in a circuit like this:



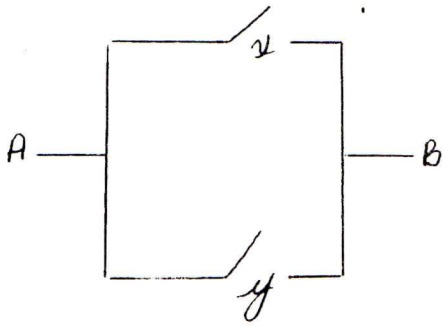
Power source of electricity, hereafter omitted for all diagrams

The switch is normally open, (as shown), but may be closed. It therefore has 2 states. We shall construct a mathematical model of switching circuits, based on an algebraic system called Boolean Algebra. We represent the switches by variables and allow the variables to take only 2 values, corresponding to the 2 states of the switch. $x = 0$ represents an open switch and $x = 1$ represents a closed switch. In the illustration, $x = 0$.

3. From your work in sets you are familiar with certain operations on sets. Each of these may be used to represent physical combinations of switches.

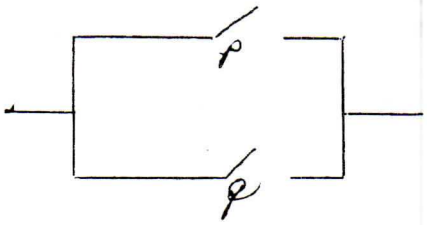
a. $x \cap y$ Called an AND Circuit. Current flows only if x and y are closed. Also called switches in series.

b. $x \cup y$ Called an OR Circuit. Current flows if x or y are closed Also called switches in parallel.



c. $\sim x$ Called not x. This cannot be easily represented. It means that if $x = 1$, $\sim x = 0$, and vice versa. There are electrical devices that will release a current pulse when the power source current is OFF and will not pass a pulse when the power source is ON.

4. We can also connect circuits. For instance, let $p = x \cap y$ and $q = x \cup y$. Then the combination p U q looks like this:



These are termed mathematical switches, as distinct from actual physical switches. What would the actual circuit look like, if p and q are defined as above?

5. Circuits are called equivalent (symbol =) if their values are equal when the same constants are substituted for the common variables in both circuits. For instance if 2 circuits A and B are both defined in terms of x and y, and A = 1 and B = 1 when x = 1 and y = 0, then A = B.

6. As an algebraic system Boolean Algebra has a fundamental set of laws. Some of these are listed below, without proof. For proof and derivation, see Dorn and Greenberg, Mathematics and Computing, in the Math Dept. library.

1. $x \cap 0 = 0$
2. $x \cap 1 = x$
3. $x \cap x = x$
4. $x \cup 0 = x$
5. $x \cup 1 = 1$
6. $x \cup x = x$
7. $\sim (\sim x) = x$
8. $x \cap y = y \cap x$ Commutative Law
9. $x \cup y = y \cup x$ Commutative Law
10. $x \cap (y \cap z) = (x \cap y) \cap z = x \cap y \cap z$ Assoc Law
11. $x \cup (y \cup z) = (x \cup y) \cup z = x \cup y \cup z$ Assoc Law

E. Exercises:

1.

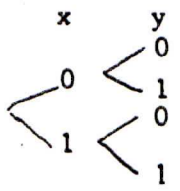
2. Draw circiuts illustrating laws (1) to (6) and (8) to (11).

52. Boolean Algebra and Truth Tables

1. Additional laws (continued from 47. above)

12.	$x \cap (y \cup z) = (x \cap y) \cup (x \cap z)$	Distributive Law
13.	$x \cup (y \cap z) = (x \cup y) \cap (x \cup z)$	
14.	$x \cup \sim x = 1$	Complementation Law
15.	$x \cap \sim x = 0$	
16.	$\sim x \cup \sim y = \sim(x \cap y)$	Dualization Law
17.	$\sim x \cap \sim y = \sim(x \cup y)$	

2. Will current flow or not for any given settings of the switches? Consider an AND circuit. $(x \cap y)$. How many possible settings of the switches can be made? You may see at once that the answer is 4. For those who do not, and for more complex circuits, we may turn to a <u>tree diagram</u>:



Follow each branch from the trunk to the tip of the branch. Each branch is a possible setting. There are 4 branches, therefore, there are 4 po sible settings if we organize these in a table and add a column for the output, we have a Truth Table. The output is filled out from the laws above, or from inspection of the physical circuits.
Truth Table for the AND circuit:

x	y	$x \cap y$ (output)
0	0	
0	1	
1	0	
1	1	

3. Other useful truth tables include:

a. Incluseive OR circuit

x	y	$x \cup y$

b.

x	y	$x \cup y$	$x \cap (x \cup y)$

c.

x	y	$x \cap y$	$x \cup (x \cap y)$

4. We can now define equivalent circuits as circuits whose truth tables are equal in the output column. What is your conclusion about 3b. and 3c. above?

5. Exercises
Fill in the truth tables above and answer question in 4. above.

53 Truth Tables by Computer

1. Using the truth tables shown above, we can make truth tables for more complex circuits. What will be the truth table for $A = x \cap y \cap z$? Use a tree diagram to find out how many <u>different</u> rows can be made.

x y z

2. Place parentheses to work with 2 variables (columns) at a time, for instance x and y. From the truth tables above, write an output column:

x y $(x \cap y)$

3. Combine this output column with the other variable (z) and use the AND truth table to get the output column for the entire circuit:

$(x \cap y)$ z $(x \cap y) \cap z$

4. Rewrite the truth table, showing the inputs for x, y, and z, but eliminating the column $x \cap y$, which was only termporarily used. In the final result we are not interested in $x \cap y$.

x y z $x \cap y \cap z$

5. As our variables increase in number, so the truth table increases in size and number of operations necessary. We turn to the computer for assistance. Here are the equivalents of some elementary circuits:

Algebra	FOCAL
$z = {\sim}x$	Z = 1 - X
$z = x \cap y$	Z = X * Y
$z = x \cup y$	Z = X + Y

Why don t we use -X for ${\sim}X$?
In the last one above we will have difficulty with X = 1 and Y = 1. Z then = 2, which is not permissible. We must include statements to change the 2 to a 1.

```
1  SET Z = X + Y
2  IF (z) .4,  .4,  .3
3  SET Z = 1
4  continuation of program
```

Of course the program does not include input or output statements, which must be supplied for a complete program.

5. Exercise

Write a complete program to print the truth table for z = x U y. Use nested DO statements to generate the input.

§4. More on computed truth tables

1. To analyze more complex circuits we use a combination of 2 general methods. The first is the nesting of loops (usually DO statements). If we have 3 variables, we will have 3 loops, 4 variables call for 4 loops, etc. The second method is to set up temporary variables, usually indicated by parentheses. For instance, if we wish to program S = x∩(y U z) we compute TEMP = y U z and then compute x∩TEMP. We need not print TEMP, although we might print it as a check or aid in debugging. The following program will compute S:

```
1.1  TYPE ! , "  X   AND (Y OR Z) = S", ! , " X      Y      Z      S "
1.2  FOR X = 0, 1; DO 2.1
1.3  QUIT

2.1  FOR Y = 0,1; DO 3.1

3.1  FOR Z = 0,1; DO 4.0

4.1  SET Temp = Y + Z
4.2  IF (TEMP) 4.4,4.4,4.3
4.3  SET TEMP = 1
4.   SET S = TEMP * X
4.5  TYPE %1.0,! ,X, "    ", Y , "    " , Z, "    ",S
```

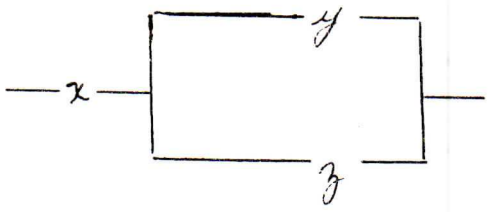
2. If no parenthese appear in the original expression, we need not use TEMP variables. For instance S = x∩u∩z can be programmed (in part) as follows: (output statements omitted)

```
1.1  FOR X = 0,1 ; D O 2.0
1.2  QUIT

2.1  FOR Y = 0,1; DO 3.1

3.1  FOR Z = 0,1; SET S = X * Y * Z
```

3. Exercise
Write a complete program to compute the truth table for S = x∩(x U y)

55. Analysis of Circuits

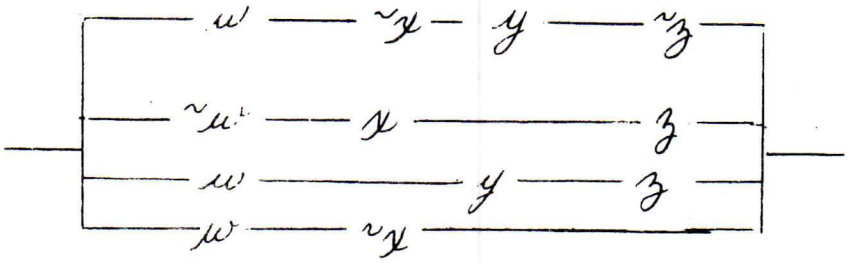
1. A circuit is analyzed by inspection of its truth table. As we saw before, it is possible to translate any circuit into a Boolean algebra expression. Consider the following example:



Translating this gives S = x ∩ (y U z). This was the example in 50. above. Its truth table is:

x	y	z	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Consider the circuit below



If we designate the top branch by C1, the 2nd by 02, etc. we can then write C = c1 U C2 U C3 U C4

2. Exercises

a. Write the equations for C1, C2, C3, and C4.

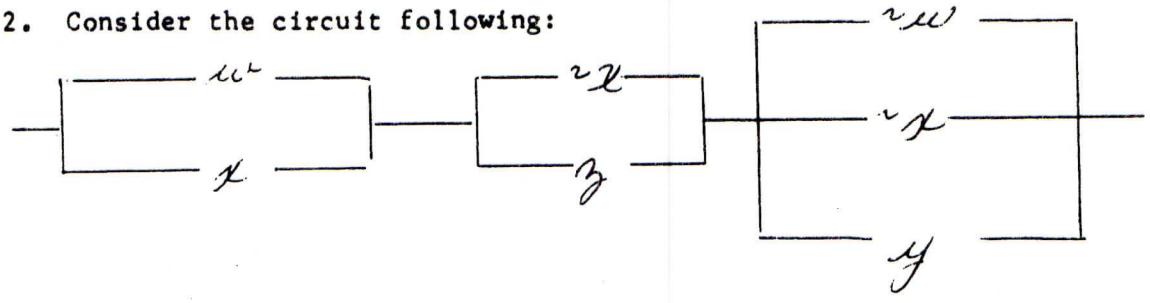
b. Write separate FOCAL programs to evaluate C1, C2, C3, and C4. Omit output statements.

c. Write a FOCAL program to evaluate C = C1 U C2 U C3 U C4. Include output statements. Assume that C1, C2, C3, and C4 have <u>already been entered</u> into the symbol table.

56. Analysis of Circuits

1. The circuit in 55 above uses 12 switches (although only 4 are independent). Is it possible to get the same output with fewer switches? We must first get the truth table for the circuit by constructing a program for it. Combine the answers to the exercises in 55 above and you will have a program.

2. Consider the circuit following:



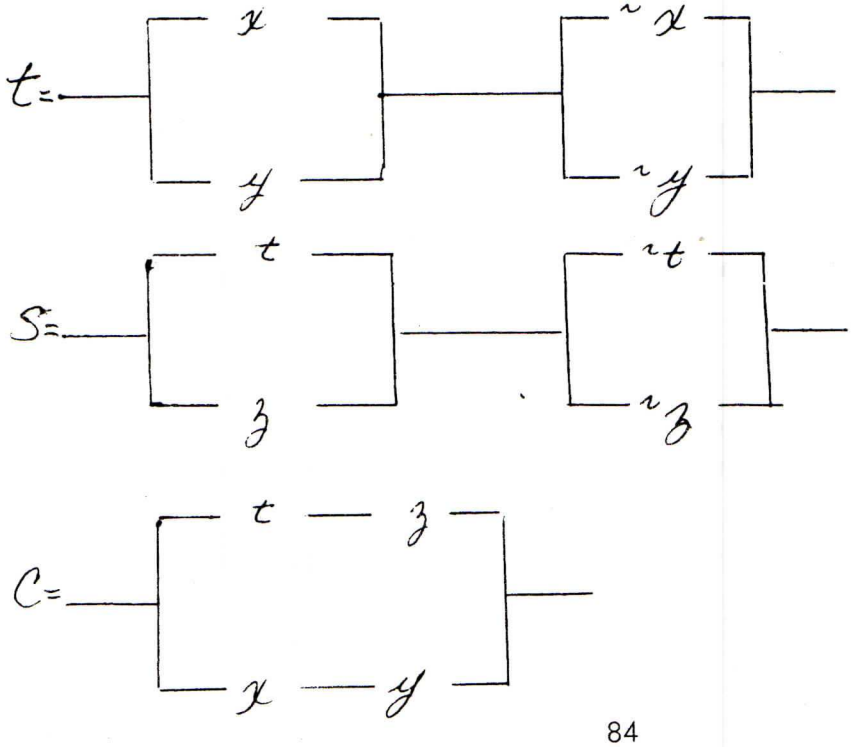
This has 7 switches. If it has the same truth table as 55, then it is an equivalent circuit.

3. Exercises

a. Write a program for the circuit in 55.
b. Write a program for the circuit in 2. above.

57. Review of circuit analysis

1. Find truth tables for the following circuits:



2. Combine the results into a single truth table, omitting t. Since C and S are independent, your table will have 8 rows, not 16. The same x and y and z columns should be used for both C and S

x	y	z	C	S

5. Circuit Design

1. In the preceding pages we learned how to find the output (expressed as a truth table) of a given circuit. The next problem is to start with the no. of switches and the desired output (expressed as a truth table) and design a circuit that will give the desired output. We will start with the following truth table, for which we are to design a circuit. It is the Exclusive OR table.

x	y	Z (output)
0	0	0
0	1	1
1	0	1
1	1	0

2. Write the truth tables for the following:

$A = x \cap \sim y$

x	y	A

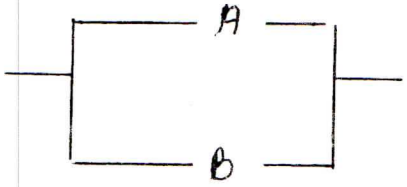
$B = \sim x \cap y$

x	y	B

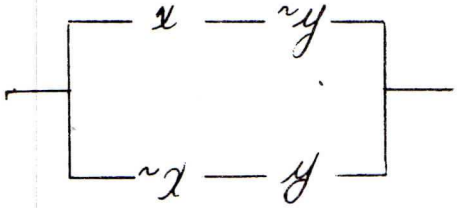
3. Write a truth table for $Z = A \cup B$, where A and B are defined as in 2. above.

A	B	Z

4. Note that the above has the output of the table in 1. above. The circuit in terms of A and B is therefore:



Substituting the equivalent circuits for the mathematical switches A and B, we get the following circuit, which is the one we were to design:



This probably seemed like pulling a rabbit out of a hat. Why did we choose the circuits that we did for A and B? There were many other possible circuits that might have been chosen (but which would not have given the correct output).

5. Rules for design of circuits from a truth table

a. Find the outputs that are = 1.

b. In rows where the output is 1, form an AND circuit of the switches. If the switch in the table is = 1, use the switch as is. If the switch = 0, use the negation ( ) of the switch.

c. The number of AND circuits will be equal to the number of 1's in the output colmn of the truth table.

d. Form an OR circuit of all the AND circuits

6. Exercise

Design 4 circuits for the following truth tables:

1.

x	y	Z
0	0	1
0	1	0
1	0	1
1	1	1

2. NOR (NOT — OR) table

x	y	$x \downarrow y$
0	0	1
0	1	0
1	0	0
1	1	0

3.

x	y	z	A (output)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

4. NAND (NOT-AND) table

x	y	$x \mid y$
0	0	1
0	1	1
1	0	1
1	1	0

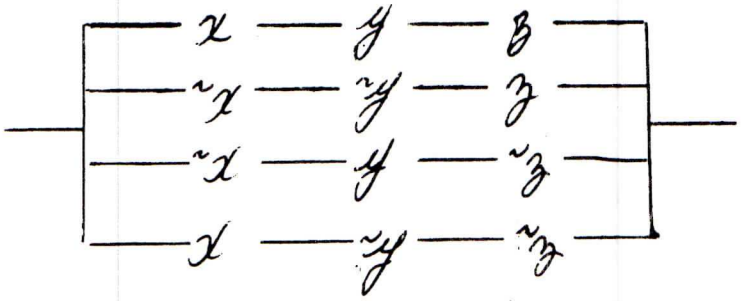
5.

x	y	z	S (output)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

59. Computer design of a one place adder

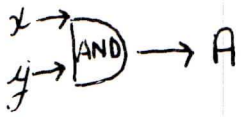
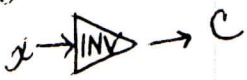
1. Look at the answer to 57.2. As some of you noticed, the C and S columns are formed from a binary table consisting of 0, 1, 10, and 11. This is analagous to a computer with 3 one-bit addresses and a one-bit accumulator and one-bit link. We can TAD any or all of the 3 addresses.

2. Let us consider the circuit for the accumulator (sum digit)



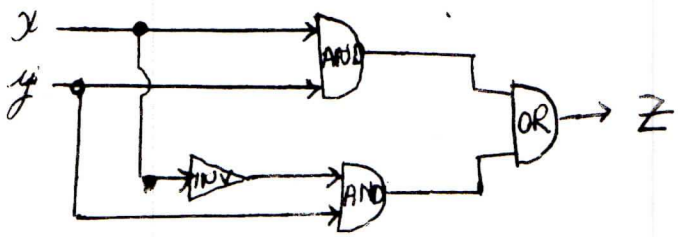
Obviously we cannot have a midget running around in the computer throwing single-ple switches. This is done electronically by means of gates.

3. We will use the following gates:

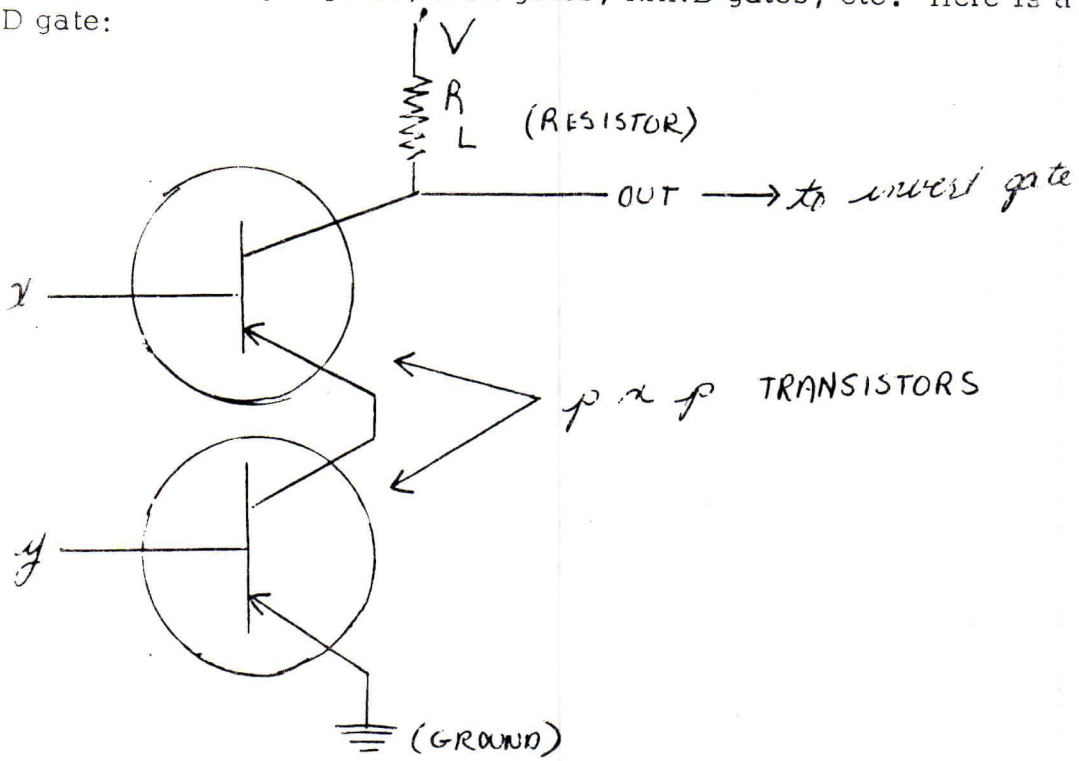
GATE	Boolean Expression
	$A = x \cap y$
	$B = x \cup y$
	$C = \sim x$

4. Note that we are limiting ourselves to 2 inputs into AND and OR gates and 1 input into an INVERTER. Gates do exist with more than 2 inputs, but we shall not use them. Also note that we are now indicating the direction of the current.

5. Using the above logical symbols, the expression $Z = (x \cap y) \cup (\sim X \cap Y)$ becomes:



6. Each of the gates has electronic equivalents. There are also electronic equivalents for 3 input gates, NOR gates, NAND gates, etc. Here is a typical AND gate:



A negative voltage (relative to ground) is introduced at V. Current flows from ground to V and also to OUT. OUT is positive. Current flows through a pnp transistor only when the inputs are negative. We define negative inputs as 1. We define positive inputs as 0. Now let either x or y be positive (0). Current cannot flow from ground to V, therefore it flows from OUT to V. This makes OUT negative (1). If you write the truth table, you will see that it is the opposite of the AND table. OUT is 0 if and only if both x and y are 1. Add an invertor gate to OUT. This makes OUT = 1 if and only if both x and y are 1. Without the invertor this is a NAND gate. The output can serve as the input to another gate. Using transistors in parallel will give an AND gate without the need for the invertor, but the circuit is not as easy to follow.

The definitions of negative = 1 and positive = 0 may seem odd, but they are widely used in computer circuits, including our PDP8/S.

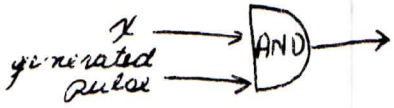
7. Exercise

a. Draw the logical circuits (using gates) for the Sum digit (S) in 57 above.

b. Draw the logical circuit for the Link digit (C) in 57 above.

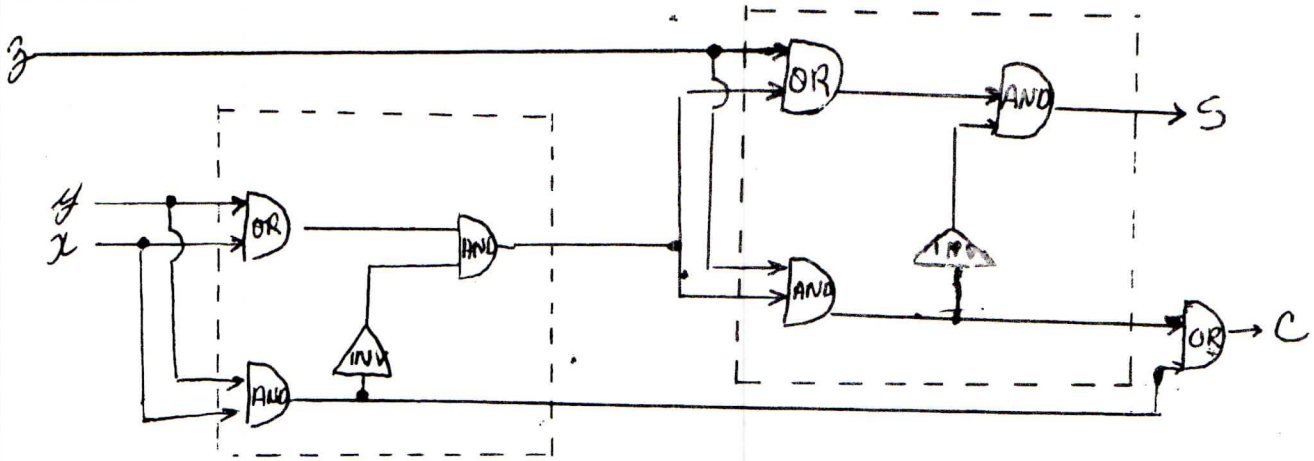
60. More on Computer Design

1. Current is carried forward in the logic circuits by alternating pulses every 1.5 micro-seconds (.0000015 seconds). This is done by introducing timing inputs at necessary points. Perhaps the simplest example is in the inputs. An input gate might look like this: (x would be a small direct current)

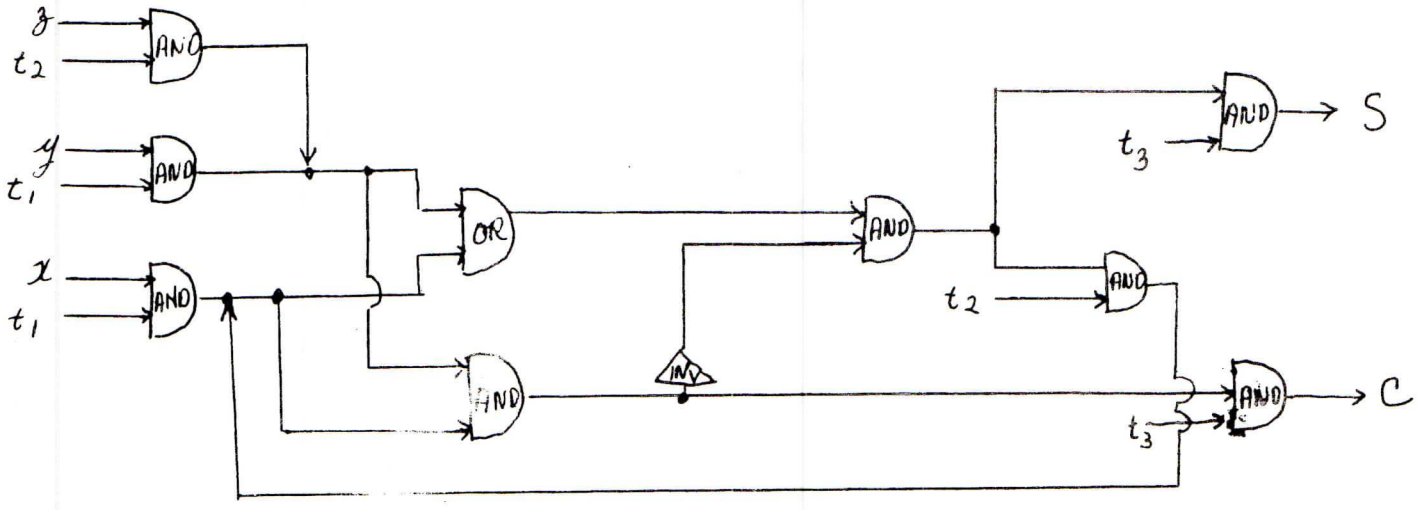


2. Optimal design

a. If we add together the link and accumulator circuits from 56. above, we will have a complete one place adder (for 3 numbers) that uses 11 AND gates, 6 OR gates, and 3 INVERTERS. By means beyond the scope of this course, the following circuit could be derived that would do the same job with 4 ANDs, 3 ORs, and 2 INVERTERS:



b. Suppose we wish to save gates (hardware). Note that the circuits within the dotted lines are the Same circuits. By the use of timing, we may use the same circuit twice and save hardware. In the diagram below, t_1 represents a pulse at Event Time 1, t_2 at Event Time 2, and t_3 at Event Time 3. Recall that our PDP/8S has 4 Event Times.



C and S must be out putted <u>after</u> t_2 or a false value may go forward at t_2.

c. As is obvious, we have saved hardware, but lost time. Suppose we wished to add two 3 bit addresses. This can be done by using 3 of the circuits in a. above in series (with minor modifications). This would require 12 ANDs, 9 ORs, and 6 INVERTERS, and would take 1.5 micro-seconds. If we decided to save hardware, we could use the circuit in b. above 3 times. This would use 5 ANDs, 1 OR, and 1 INVERTER, but would take at least 13.5 micro-seconds. This conflict between hardware and speed is one of the basic facts of life for computer design.

3. Exercise

Construct a logical circuit for the following truth table:

x	y	z	S
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

61. Mathematical Probability

1. Probability deals with events whose outcomes are not certain, such as tessing a coin. The result of an experiment i or trial, is called a simple event, or sample point. Heads is a sample point for the toss of a single coin. The collection of all sample points is called a sample space. The sample space for the single coin toss is H, T.

2. Let us toss 2 coins, one after the other. There are 4 sample points, namely TH, TT. We consider TH and HT as 2 distinct points, for we are to limit ourselves to "likely to occur" points. If we consider TH and HT as one point, it is more likely to get HH or TT.

3. If we define the sample space as a set, then any subset of the sample space defines an event. An event that contains only one sample point is a simple event. An event that contains more than one point is an event. Some events include (in the 2 coin tossing experiment):

a. At least one head
b. A head is first
c. Two heads (a simple event)

4. The operations on sets, namely $\cap$, U, and $\sim$ are applicable. Venn diagrams are also helpful. If A is defined as at least one head and B is defined as a head is first, what

5. Two definitions not made in Boolean algebra must be made:

a. E = Sure Set and contains all events.
b. $\emptyset$ = null or Empty Set and contains no events.

6. The cardinal number of a set or subset is the number of sample points in the set or subset. It is expressed as n(A). How many points are there in n(A U B)?

7. From the definitions above, we can derive the following equations:
$n(A \cup B) = n(A) + n(B) - n(A \cap B)$. If the two sets have no points in common, the equation simplifies to $n(A \cup B) = n(A) + n(B)$

8. The Law of Large Numbers states that if an experiment is performed many times, then the number of times an event occurs, divided by the total number of experiments, will be close to a constant value. For instance, if we toss a single coin 1000 times, the number of heads divided by 1000 will be close to 1/2. This fraction is called the probability P(H) of an event H. Another way of stating this is to say that if we repeat an experiment many times, a fraction P(H) of the experiments will produce the event H. 1000 tosses of a coin should produce close to 1/2 of 1000 = 500 heads. Some of these terms are admittedly vague, but a more precise definition cannot be given here.

9. The following laws and definitions will be useful:

$0 \leq P(x) \leq 1$

$P(a) + P(b) + P(c) \ldots\ldots = 1$, where a, b, c, ... include all the sample p points.

$P(x) = \frac{n(x)}{n(E)}$

10. Exercises

a. What is the probability of the following events:

1. a 3 on one toss of a die
2. An even number on one toss of a die
3. An ace in one draw from a 52 card deck
4. A spade in one draw from a 52 card deck
5. You receiving an A, figured on the first day of the term
6. You receiving an A, figured as of now

b. A jar contains 2 red and 2 white balls. A ball is drawn and then another, after replacing the first one. What is the probability that at least one red ball is drawn?

c. That a black ball is drawn?

d. After 3 draws without replacement, that a red ball is drawn?

62. More on Probability

1. In exercise b. of 61. above, we are concerned with the probability of a red on the first OR second draw. We worked it out with a tree diagram. It can also be worked out with formulas, with $\cap$ = AND and U = OR.

$$P(A \cap B) = P(A) * P(B)$$
$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

These do not apply when all events are not equoly likely. They also do not apply when the outcome of B depnds on A, or vice versa. For instance in exercise bof 58. above, if we do not replace the first draw, the second probability will depend on what was drawn first. Also note that sometimes $P(A \cap B)$ does not make sense (What is $P(H \cap T)$ for one toss of a coint?) or = 0.

2. $P(A \cap B)$ can be extended to more than two events. $P(A \cap B \cap C \ldots) \tilde{=} P(A) * P(B) * P(C) \ldots$ The other formula cannot.

3. The mathematical analysis of probability (drastically shortened) that we have studied thus far can be used for simple problems. As our problems become more complicated and as the sample space increases in size, the formulas become difficult to use. Another approach is to carry out a large number of experiments and tabulate the results. For instance, what is your probability of winning at dice? One method would be to toss the dice for 1000 games and record the number of wins and losses. This technique of simulating many experiments whose outcomes depend on chance (are not certain) is called the Monte Carlo Method and was developed as recently as 1944 during work on the Atomic Bomb.

4. The most basic idea in Monte Carlo methods is that of random numbers. This means not only that any number is equally likely to occur, but also that there should be no pattern in the order in which the numbers appear. The size of the numbers does not matter. If we limit ourselves to the digits 1 to 6 (as in a single die), they will obviously repeat a sequence at some point. However, unless they repeat the entire sequence prior to that point, all are still random numbers. For instance, if the numbers 561243 start off the sequence, they will reappear at some point in that order. If all the sequence between the first 561243 and the second 561243 repeats after the second appearance, then only the part from the first digit to the reappearance of the first digit is random. If the entire sequence does not repeat, then all are random numbers.

561243123456789561243123456789... only first 15 are random
561243142365348756124354437458834...All are random.

5. Exercise

a. Suppose you had a sequence of 3 digit random numbers. How could you use this to play dice, where only the numbers 1 to 6 are wanted.

6. Random Numbers

1. Here is a sequence of 100 3 digit random numbers printed by our random number generator (FRAN). To use them to simulate a game of dice, we might divide each of them by 6, add 1 to the remainder, and use that as the roll of one die. Ignore the + or -.

2. Verify that you would win on your first 3 rolls of the dice, that thereafter you would have the point 8, and 6 rolls thereafter you would lose.

0.250 0.624 0.691 0.002 0.005 0.031 0.375 0.666 0.060 0.562
0.833 0.005 0.017 0.063 0.853 0.000 0.688 0.085 0.031 0.063
0.008 0.062 0.813 0.000 0.007 0.093 0.001 0.001 0.094 0.000
0.000 0.231 0.250 0.624 0.000 0.000 0.000 0.000 0.000 0.068
0.002 0.265 0.313 0.625 0.029 0.000 0.003 0.503 0.020 0.001
0.086 0.109 0.068 0.066 0.067 0.690 0.045 0.031 0.069 0.004
0.002 0.004 0.031 0.500 0.438 0.665 0.828 0.833 0.838 0.516
0.165 0.431 0.156 0.170 0.178 0.594 0.773 0.363 0.228 0.656
0.652 0.633 0.096 0.579 0.624 0.130 0.419 0.425 0.524 0.237
0.220 0.736 0.728 0.731 0.400 0.053 0.699 0.248 0.001 0.854

3. Exercise:

a. Use the sequence above to simulate Black Jack. Do not write a FOCAL program. Generally the rules are that ace = 1 or 11 (your choice), honors 10, all other are face value. You draw 2 cards. If they = 21, you win. If < 21, you draw cards to get as close to 21 as possible. If you go over 21, you lose. Then the dealer plays. If he ties you, he wins. If he is closer to 21, he wins. If he goes over 21, or you are closer to 21 than him, he loses.

b. Write a FOCAL program to simulate the playing of dice, without keeping track of bets. In other words, it would only print WIN or LOSE.

c. How would you derive the numbers 2 to 12 from the above sequence? Would this simulate a dice game?

6. Random Number Generator

1. In 6B. above we gave some fundamental ideas on random numbers. We use the function FRAN. This function derives the numbers by an algebraic algorithm. If we know the algorithm, we could repeat any sequence. Furthermore, at some point FRAN will repeat itself. Therefore, it is a pseudo-random number generator, since it is computed as opposed to recording rolls of dice or results of a roulette wheel. It will be instructive to construct our own pseudo-random number generator. Hereafter the prefix pseudo will be omitted, although it still applies

2. A cycle is the sequence of random numbers before the sequence repeats. Cycle length is the number of numbers in the cycle. Let us take any integer from 1 to 10 and add it to itself. If the result $<$ 11, use it. If $>$ 11, subtract 11 from the result. Do this for 15 numbers:

This is a sequence of random numbers. What is its cycle length?

3. The above method can be used to give longer cycles. If we use 13 instead of 11, we will get a cycle length of 12. However, all numbers cannot be used in this way. The number must be prime. It must also have a primitive root of 2, which will not be explained. Here is a partial list of prime numbers which have a primitive root of 2:
3, 5, 11, 13, 19, 29, 37, 53, 59, 61, 67, 83, 941, 947

4. Another method of increasing the cycle length is to add 2 sequences. This will give us numbers between 2 and the sum of the largest possible numbers in the sequences. Its cycle length will be the lowest common multiple of the 2 cycle lengths. For example, derive a sequence from the prime 13:

Now derive a new sequence from their sum. Carry this out for 15 numbers only. What is the cycle length?

Sequence 1	Sequence 2	Sum

5. Exercise

a. Write a FOCAL program to generate a series of p-1 random numbers based on the prime number p.

65. More on Random Number Generators

1. In 61. above we wrote a FOCAL program to generate a random number sequence. How random is it? It cannot be truly random, being a pseudo-random generator. There are tests that will tell us whether a sequence may be used instead of a true random number sequence.

2. There are 8 tests that should be met. We will not list all of them. They were derived from college level calculus and statistics. The easiest to apply is to test the total of odd and even numbers. The numbers should be half odd and half even. Another is the gap test. We count the intervals in digits between the reappearance of a single digit. An average of these intervals should be close to 10. This should be done for all digits 0 to 9. Both these tests should be run over the complete cycle.

3. The D^2 test is a test of the numbers rather than the individual digits. It is based on the distance formula for the distance between 2 points. If we express each number as a decimal between 0 and 1 and take each 2 successive numbers to represent a Cartesian pair (x,y), then use the formula $D^2 = (x_1 - x_2)^2 + (y_1 - y_2)^2$, statistically the values for D^2 should be < 2.0. A complete table of probabilities for all values from 0.1 to 2.0 has been compiled. Here is part of that table:

D^2	Probability (cumulative)
0.1	0.234832
0.2	0.409805
0.3	0.549300
0.4	0.662018
0.5	0.752987
0.6	0.825601
0.7	0.882349
0.8	0.925163
0.9	0.955593
1.0	0.974926

4. Exercise

a. Write a FOCAL program to calculate a sequence based on adding together 2 sequences derived from the primes p and q, as mentioned in 60. above.

b. Use the D^2 test to test the 60 numbers that you get from using the above program to compute a sequence based on the primes 11 and 13.

66. Computer Modelling

1. The earliest example of a mathematical model is the Pythagorean Tehorem (6th Century B.C.). Later examples include Newton's Laws of Motion and Einstein's Theory of Relativity. Any formula or set of formulas that are used to portray a physical process, object, or observable phenomenom is a model. In the field of statistics and probability we have the basic formulas of Bernoulli (18th Century) and Pearson (19th Century). One of the first computer applications, besides the programming of simple formulas, was an air battle simulator designed in the early 1950's at JohnsHopkins. In this a long series of formulas were used to describe a battle between 2 opposing sides. These formulas were then programmed into a computer. Since that time, the field has expanded tremendously.

2. There are 3 types of models currently in use:

a. Deterministic - an analytic representation in which there are unique outcomes for a given set of inputs. A simple example is a formula such as Hero's Formula.

b. Stochastic - functional relationships depend on chance parameters. Example--any use of Monte Carlo methods.

c. Expected value - one in which expected values or means are assigned to the chance parameters, possible modified by dispersion measures such as the standard deviation, probable error, etc.

3. In **62.** above, we noted that Monte Carlo methods may be used to determine probabilities when we choose not to (or cannot) determine them by mathematica analysis. What is the probability of winning at dice if you hold (toss) the dice? This can be determined analytically by an examination of the sample space. However, let us determine it by Monte Carlo methods.

4. Our program in **63** above is a computer model. It simulates by logical mathematical methods the game of dice. We can use it, with slight modifications, to determine the advantage or disadvantage of holding the dice. If the probability of winning is greater than losing, obviously the advantage lies with the holder. If the probability of losing is greater, the advantage lies with the players who bet against the holder.

5. Exercise

a. Modify the program in **63** to keep a tally of the number of times you win or lose over a run of 500 games. There should be no printout except for the final tally of wins and losses.

b. Run the program

c. Run the same program <u>after removing</u> the loss on a roll of 3 on the initial roll. How do the results compare?

67. More on Modelling

1. We shall attempt a model of a basketball game. It is not expected that our model will be completely reliable since we must cut many corners in order to keep our program within workable limits. One reason for selection of basketball is that the game is relatively simple in scoring and in number of variables. Consider the possibilities on a shot. Either you make it or you don't. Compare this with the possible outcomes of a time at bat in baseball.

2. The probabilities will be based on past performance. This is always a good guide to future performance, but is not infallible. The long history of upsets in all sports proves this. In addition, we will base our data on a single game, that of 3 May 69 between Boston and L.A. This is a ridiculously small sample, but the season figures for any team and its opponents are not available to us. The method to be used for an entire season would be the same, except that the totals would be larger.

3. In all our decisions we will compute probabilities where events E_1, E_2, E_3... are assigned probabilities $p_1 + P_2 + P_3 + \ldots = 1$. We then generate a random number U between 0 and 1. E_1 occurs if $U \leq P_1$. E_2 occurs if $P_1 < U \leq P_1 + P_2$. E_3 occurs if $P_1 + P_2 < U \leq P_1 + P_2 + P_3$, etc.

	G	ST	F	FT	R	A	PERS
BOSTON							
Bryant	7	18	4	4	9	5	4
Havlicek	8	26	3	4	9	2	4
Howell	1	10	2	3	3	1	6
Jones	4	8	1	1	2	0	4
Nelson	10	19	5	5	9	1	4
Russell	3	8	3	3	19	2	5
Sanders	1	1	1	1	2	0	4
Siegfried	4	14	4	4	5	3	2
L.A.							
Baylor	9	18	8	10	10	0	4
Chamberlain	1	5	6	10	18	4	3
Counts	5	9	4	6	5	0	5
Egan	3	14	1	1	0	2	4
Erickson	2	6	3	7	10	0	2
Hawkins	1	3	0	3	6	1	2
West	9	19	8	11	4	3	4

5. Exercise

Write a flow chart to simulate the sequence of events from the center tap ($p = 1/2$) until the team either makes its basket or misses. Remember that <u>all</u> possibilities must be included. Ignore the movement of the ball toward the basket. Consider only who shoots and whether he makes it. Also foul or not. At each event, there should be a decision box and a probability assigned.

APPENDIX 1
The FORTRAN language

1 .Compilation of a FORTRAN program

a. We will investigate entry of FORTRAN programs into the computer. The FORTRAN language is obviously not machine language. A program has been previously prepared that translates FORTRAN statements into machine language. For example, GO TO is 5xxx in machine language. Similarly the symbol must be translated into more than one instruction, for the computer multiplies by repeated addition. This translation program is called the FORTRAN Compiler.

b. Basically the computer is designed to accept all input from the SR. Entering programs by this method would be very tedious. We must enter a program that allows the computer to accept input from the high or low speed reader. This program is called the RIM (Read In Mode) Loader. It comes in 2 versions - one for the high speed reader and one for the low. The RIM Loader must be entered by hand, using the SR. Once entered, it will not be destroyed, except by unusual errors by a programmer. DO NOT USE the addresses listed below if at all possible, except for the RIM Loader. The RIM Loader should always be checked, for the previous user might have been using a different reader. The main use of the RIM Loader is to load the BIN Loader. The high speed RIM Loader follows:

ADDRESS	CONTENT
7756	6014
7757	6011
7760	5357
7761	6016
7762	7106
7763	7006
7764	7510
7765	5374
7766	7006
7767	6011
7770	5367
7771	6016
7772	7420
7773	3776
7774	3376
7775	5357

c. The BIN (Binary) Loader is a program in RIM format that allows us to use BIN format. Since BIN format is much shorter than RIM format, most DEC tapes are in BIN format. The BIN Loader should be entered via the high speed reader, which means we must use the high speed RIM Loader. Once loaded, the BIN Loader will not be destroyed, except by unusual programmer errors. To load the BIN Loader:

1. Place tape in reader, with leader code over photo-cell.
2. Load address 7756
3. START

d. We use the BIN Loader to load the FORTRAN Compiler. To use the BIN Loader:

1. Place tape (compiler) in reader (high speed).
2. Load address 7777
3. Depress bit 0 (Leave up for low speed)
4. START
5. After tape stops, check AC. If AC = 0, go on with next step. If AC ≠ 0, reload compiler. If AC ≠ 0 again, notify teacher.

e. The FORTRAN Compiler is now used to compile your source FORTRAN program.

1. Place source tape in low speed reader
2. Load address 0200.
3. Set low speed reader to ON LINE.
4. Turn on reader. Turn on punch
5. START
6. The tape that is punched out is your object (machine language) program.

f. If you have any errors, they will be typed out after the compilation is finished. This means that your source program will have to be corrected, and recompiled.

g. To compile additional programs, place new source tape into reader, turn on reader and CONTINUE.

h. Exercise: Compile your program.

2.

Revised Section 3.1 -- PDP FORTRAN

3.1 The ACCEPT and FORMAT statements

If a problem is to be done only once, the data can be entered as constants. This is ordinarily not the case: programs are usually set up to read different sets of data for the same variables: the same program can then be used with as many sets of data as desired.

Data is entered into the computer by the execution of an ACCEPT statement that lists the names of the variables for which new values are to be read in. The new values must be punched in the same sequence as the variable names are listed in the ACCEPT statement. The first value on the data tape goes with the first variable name, the second value with the second variable name, etc. The execution of an ACCEPT statement always initiates the reading of a new line on the data tape. If, for instance, there are 6 data values punched on one line, they cannot be entered with 2 ACCEPT statements. The only way to enter the 6 values from one line is to provide an ACCEPT statement that lists the names of all 6 variables.

So far we have discussed how to provide 3 items of information about the input operation:

1. What data is to be used and whether this is an input or output operation.
2. Which variables in the program are to receive new values.
3. The order in which the values appear on the data tape, This is specified by the order in which their names appear in the ACCEPT statement.

There is one more item of information required: in what format are the data values? Is the data fixed or floating or both?

This information is provided by a FORMAT statement. A complete discussion is deferred to Chapter 7, but we can get the basic ideas now. Suppose that it is desired to read new values of 3 variables named A, I, and X. The values are punched on a data tape as follows:

```
21.4E10 16 1.98
```

Note that one or more spaces are used to separate different data words.

The data words could be read into the computer by the following 2 lines excerpted from the program:

```
      ACCEPT 69,A,I,X
69;FORMAT (E,I,E)
```

The 69 in front of the FORMAT statement is its statement number; this number was chosen arbitrarily by the programmer. More about this in Chapter 4. For the present, limit your program numbers to 2 digits. Here we use a statement number to identify the FORMAT statement that is associated with the ACCEPT statement. All ACCEPT statements must refer to a FORMAT statement.

3 3.2 Summary of FORTRAN compilation and execution

To make use of a computer using FORTRAN, it is vital to understand the relation between the source and object programs and how input and output operations tie in. We may summarize the procedure as follows:

1. The program is written on paper by hand by the programmer.
2. The program is punched onto tape. The result is the source program tape. It does not include data. The data words are punched onto a separate data tape, usually called input tape.
3. The FORTRAN compiler (a tape supplied by DEC), which is a large program of machine instructions, is read into the computer.
4. The FORTRAN compiler reads the source program tape into the computer, compiles (translates) the source program into machine language, and punches the machine instructions onto another tape called the object program tape. The object program has not been executed yet and the data tape has not yet been read.
5. The FORTRAN OPERATING SYSTEM is read into the computer. The COMPILER only translates the source program. The OPERATING SYSTEM, also a tape from DEC, executes the object program.
6. The data tape is entered and (we hope) the correct answer(s) appears.

3.3 The TYPE STATEMENT

1. In order to have our answer(s) typed on the teletype we must use a TYPE statement. This is used exactly the same as the ACCEPT statement. For instance:

```
TYPE 40, AREA, PERM
40; FORMAT (E)
```

These 2 lines would cause the values of AREA and PERM (calculated by the computer) to be printed on the teletype in E format.

2. Text is outputted by use of ". For example:

```
TYPE 16
10; FORMAT ("HELLO")
```

4. Debugging a program

a. After compilation, diagnostics may be printed on the ASR 33. For this reason the ASR 33 must always be ON LINE during compilation. Generally the compiler will detect technical errors, such as illegal characters, references to non-existent statement numbers, etc. It will not detect logical errors, such as infinite loops.

b. The diagnostic messages are in this format 0005 10 11. The first 4 digits are the statement number of the last numbered statement before the erroneous statement (not the erroneous statement itself); the second 2 digits represent the octal number of statements after that numbered statement; and the last 2 digits are a code indicating the nature of the error.

c. The following is the diagnostic code:

00 Mixed fixed and floating
01 Two operators adjacent (A=B+/C)
02 Compiler error - contact teacher
03 Illegal comma
04 Too many operators in a single statement
05 Fixed function argument (A = SQTF (J))
06 Variable subscript in floating mode (A (Z))
07 More than 64 variable names in a program
10 Program too large
11 Unequal number of left and right parentheses
12 An illegal character was detected and ignored
13 Compiler is unable to recognize or process this statement
14 Program too large (same as 10)
15 A subscripted variable is used before the appearance of a DIMENSION statement, or a subscripted variable does not appear in a DIMENSION statement.
16 Statement too long (more than 128 characters on a line)
17 Floating point operand should have been fixed (DO 10 I = 1,B)
20 A statement number has been referenced that does not appear in program
21 More than 40 numbered statements in source program
22 Too many incompleted operations (6=A + (B+(C+(D+(E+...)
23 Too many statements referenced before defined
24 Use of READ or WRITE statements

d. In the example in c. above, we find that an error has occurred 8 decimal (10 octal) statements after statement 5. The error was that there were an unequal number of left and right parentheses.

e. After correcting your errors, you must compile the corrected tape as in 6. above.

f. Exercises

1. What do the following error messages indicate?

0003 06 05
0060 12 12
0077 17 03
0007 02 01
0015 15 16
program excerpts?

2. What error messages would be generated by the following:

a) 10;A=I (J+1)
B = A * (B+SQTF(C)

b) 5;C=A+B
—5 lines —
D=A (X+Y)

c) 19;X=Y+Z
— 15 lines —
A = X/N

d) 3;A=B+C
— 3 lines —
DO 10, I=1,N

e) 6;PRINT 16,X
16;FORMAT (E)

. Executing the program

a. The result of a successfully compiled program is an object tape representing your program in machine language. To run this program and get answers we must use another BIN format tape called the FORTRAN Operating System. A computer with a larger memory will have only a compiler, and the operating system will be included in the compiler.

b. Loading a compiled program
   1. Using the BIN Loader, (see 6.d, above) load the FORTRAN Operating System.
   2. Place object tape in low speed reader
   3. Turn on reader.
   4. Turn ASR 33 to ON LINE
   5. Load address 0200
   6. START
   7. If AC = 0, program is loaded correctly into core. If AC ≠ 0, reload. If AC still ≠ 0, contact teacher.

c. Executing the loaded program
   1. Be sure that SR bits 0, 1, 2 are down
   2. If input data is on tape, put tape into low speed reader. If input data is to be typed in, skip this step and step 3.
   3. Turn reader ON
   4. CONTINUE
   5. If input is to be typed, start typing.
   6. Output will appear on the ASR 33 keyboard (You hope).

d. Re-executing a loaded program
   1. Do NOT reload program
   2. Load address 0201.
   3. START

e. Miscellaneous info
   1. To execute another compiled program, start over at b. 2 above
   2. The Operating System destroys the Compiler, because it occupies many of the same addresses. To compile another program, you must start by reloading the Compiler (see 6.d. above). Obviously this will destroy the Operating System.
   3. If the program does not execute correctly on the first try (see c. above), try re-executing it (see d. above)
   4. There are many options available at various steps. These will be studied later.
   5. If your program stops during execution, see 9. below.

f. Exercise: Execute the program you compiled as an exercise in section 7. above.

6 . Debugging a program

a. Not all errors are detected by the Compiler. Some errors can only be detected by the Operating System. When such an error occurs during the running of a program, the computer types out an error message in a numerical code. The computer then tilts. If CONTINUE is pressed, the computer takes various courses of action.

CODE	ERROR	ACTION TAKEN ON CONTINUE
Tilt # 11	÷ by 0.	Quotient set to largest + or - number representable in computer, then goes on to next instruction
Tilt # 12	Floating pt exponent > ± 2047	Execute next instruction
Tilt # 13	Illegal operation code	Executes next instruction
Tilt # 14	Transfer to 0000 or 0001	No recovery possible
Tilt # 15	Non-format statement used for format	Executes next instruction
Tilt # 16	Illegal format statement constituent	Examines next constituent
Tilt # 17	Attempt to fix large floating number	No recovery possible
Tilt # 20	Attempt to take square root of - number	Takes root of absolute value
Tilt # 21	Attempt to raise - number to power	Raises absolute value to power specified
Tilt # 22	Attempt to find log of 0 or - number	Takes log of absolute value. Will not continue if 0.
31 - 34	we should not get these errors	
76 and 77	may be computer error. No recovery possible. Notify teacher	

b. Exercises:

1. What error message would result from the following statements?
   a) TYPE 16, N
      16; DO 20 I = 1, M
   b) N = X where X = 59.27E+15
   c) Z = SQTF (X) where X = -25.2
   d) Z = X-Y where Y > X
      A = SQTF (Z)
   e) Z = A-B where A = B
      Y = X/Z