3400
3600
3800

COMPUTER SYSTEMS
COMPASS

TRAINING MANUAL
VOLUME I

CONTROL DATA
CORPORATION

RECORD of REVISIONS

REVISION	NOTES
A (3-1-66)	Final Edition Released.
B (4-19-67)	Publications Change Order CA16274. Manual enlarged from one to three volumes, replacing 3600 COMPASS Programming Guide, Pub. No. 60166700, which becomes Volume 2 of the new 3-volume set. Miscellaneous corrections to Volume 2, pages 1-1, 1-21, 1-26, 2-10, 2-14, 7-3, 7-10, 11-5, 12-26, 12-30, 13-25, 15-13, 15-18, 15-22, 15-30, 15-42, 16-6, 17-2, 17-3, 17-6, 18-2, 18-6, 18-10, 18-14, 18-18, and 18-22. This edition obsoletes all previous editions.

FORM CA230 REV. 1-67

Pub No. 60166700

Printed in United States of America

Address comments concerning this manual to:

Control Data Corporation
Marketing Training Department
8100 - 34th Avenue South
Minneapolis, Minnesota 55440

FOREWORD

This manual is intended as a guide in learning how to program the upper 3000 computer systems. It includes a hardware concept of the systems, the use of the COMPASS programming language, and the use of the SCOPE monitor. Step-by-step example problems, with and without given solutions, are included to develop the capability of using the language.

This manual is a major revision to and a replacement for the 3600 Computer System COMPASS Programming Guide and retains the same publication number. It is now expanded to three volumes.

Volume I

This volume consists of three sections. The first section deals with the introduction to the systems. The second section deals with the central processor. The third section deals with problem-oriented exercises in which random instructions are picked to solve problems.

Volume II

This volume consists of one section. The instruction repertoire is divided into groups. Groups 1-18 are hardware instruction groups, and groups 19-25 are pseudo instruction groups. Each group is followed by explanations of new concepts and problems designed to use instructions from the group.

Volume I

This volume consists of two sections. The first section deals with the SCOPE system. It shows how to run jobs under the system and explains new concepts such as overlay processing and library preparation. The second section contains several computer output listings obtained as a result of running the example problems under SCOPE.

REFERENCES

3400 SCOPE / COMPASS Reference Manual	Pub. No. 60057800
3400/3600/3800 Instant TAPE SCOPE	Pub. No. 60059000
3600 Computer System Reference Manual	Pub. No. 60021300
3600 COMPASS Reference Manual	Pub. No. 60052500
3600 Instant COMPASS	Pub. No. 60056500
SCOPE Reference Manual	Pub. No. 60053300
3000 Series Peripheral Equipment Reference Manual	Pub. No. 60138800

CONTENTS

VOLUME I

SECTION I

INTRODUCTION TO THE SYSTEMS

HISTORY

The 34/36/3800 computer systems are three separate distinct computer systems in Control Data's upper 3000 line. Each of these systems is designed for large scale scientific problems and large-volume data processing. Each is built and programmed essentially the same way. Any minor difference among the systems lies in the area of expandability and timing.

Since the 3800 system is the latest and most advanced of the three systems, most of the concentration will be towards it. Occasionally reference will be made to the 3400 and 3600 systems to point out the differences.

The upper 3000 line was started with the introduction of the 3600 with production beginning around 1961. By 1966 there were 50 of these systems in the field. Within this period the other systems were introduced. The 3400 system, smaller than the 3600 and with less features, began production in 1963. About 20 of these systems have been produced. The 3800 system, slightly larger and much faster than the 3600, began production in 1964 with approximately 7 having been produced by 1966.

BASIC SYSTEMS MODULARITY

The upper 3000 systems are constructed in functional modules providing the maximum in flexibility and capability in system s applications.

Since they are constructed in functional modules it is important for the programmer, before using the system, to understand what each module does and how it operates within the system.

A basic system (e. g. 3800) consists of a central computer, an input/output section, magnetic core sotrage, and a console. The basic diagram might look like Figure 1-1.

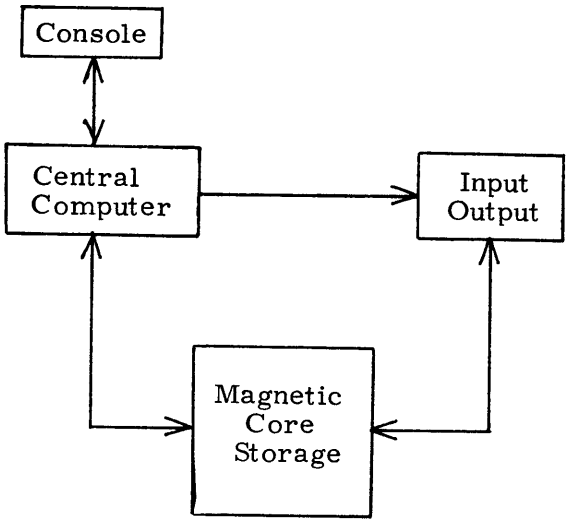


Figure 1-1

In this diagram you might note how magnetic core storage is shared by both the central computer and I/O. This is the principle of the upper 3000 line. Further explanation on this will be given.

At this time we want to take each module, one at a time, show what it looks like, and what function it performs in the system. Later we will see how it operates within the system.

CENTRAL PROCESSOR

The basic module within the system is the Central Processor; the 3804, 3604 or 3404. Some people might call it the Computation Module, Mainframe or Main Control. All are the same. Figure 1-2 shows an example of the 3600 Central Processor.

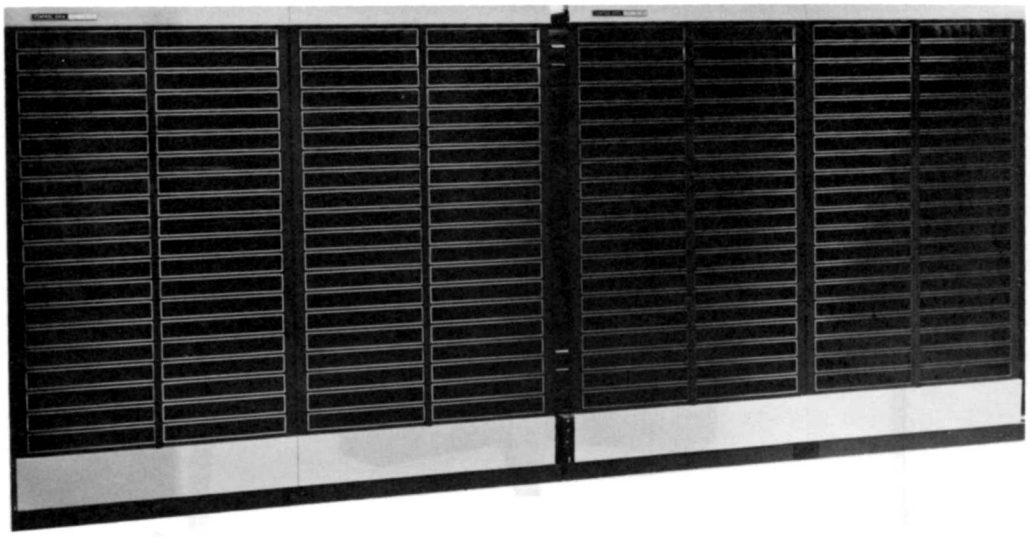


Figure 1-2

Physically the Central Processor is a huge frame about 7 feet high and 18 feet long filled with thousands of printed circuit cards. Logically it contains the circuitry for all registers and status that can be displayed on the console, all control to perform arithmetic and logical operations, and all control to service interrupts and initiate input/output operations. Logically then, this module interprets the program and performs the execution of it.

CONSOLE

The Central Processor connects directly to the Console. The consoles for the 3600/3800 systems look the same. An example is shown in Figure 1-3.

Figure 1-3

The 3801/3601 Consoles are divided into two portions. The right portion is called the Operator's Portion. It contains the typewriter where the operator can communicate with the system. It also contains status of the system, six sense switches, and three jump switches. The left portion is called the Maintenance Portion. It contains the octal display registers where the maintenance engineers can examine the contents of the registers. These registers will be shown and explained in detail later. This portion also contains a few maintenance switches.

The console for the 3400 system is shown in Figure 1-4.

Figure 1-4

This console represents only one-half of the 3400 console - the Operator's Portion. On this portion you can see the typewriter, the status display indicators, and a few of the switches.

MEMORY

The Memory for the 3400/3600/3800 systems is similar when we speak of the "bank". A bank is a memory module of 32, 768 storage addresses ranging from address 00000 through address 77777_8. Each address holds a 48-bit word (actually 48 bits plus 3 parity bits) that can be transferred to either the central processor or to peripheral equipment by means of different access lines.

The 3800 memory module is termed a 3803. Up to eight of these modules can be incorporated into a system. They would be designated as banks 0-7.

Figure 1-5 shows one 3803.

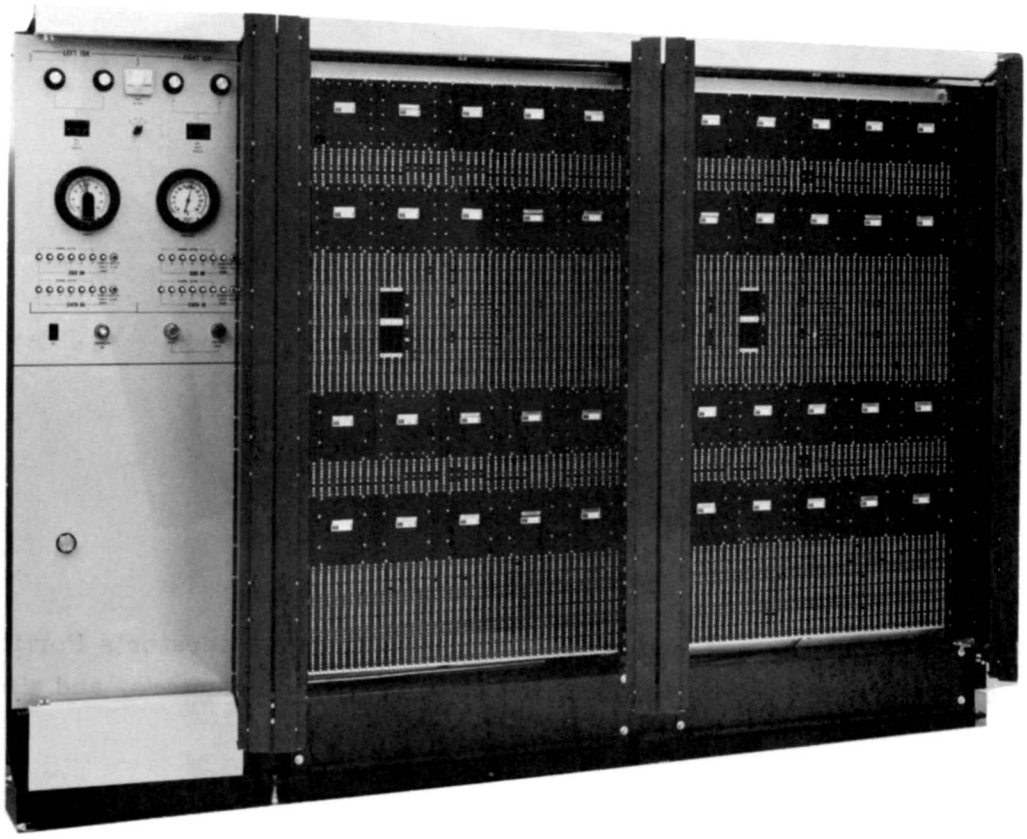


Figure 1-5

The small attached unit on the left contains a simple efficient refrigerator unit using freon to cool the module. It also contains the power supply.

FUNCTIONAL FLOW BETWEEN CENTRAL PROCESSOR AND MEMORY

At this time we would like to see how the central processor references memory for data and how data transfer takes place.

The 3600/3800 central processors, when requiring a memory reference, will generate an 18-bit address (18 bits plus 1 parity bit) and transmit it to memory. The upper three bits of the 18-bit address will reference the bank and the lower fifteen bits will reference the address within that bank. The address is accompanied by either a "read" or "write" signal. If it is a "write" signal, the central processor will also generate a 48-bit data word (48 bits plus 3 parity bits) to be stored and transmit it.

When memory receives the signals, a timing cycle is initiated, the data is read or written at the specified address and a "resume" is returned to the central processor along with a data word if the operation is a "read".

Figure 1-6 illustrates this process.

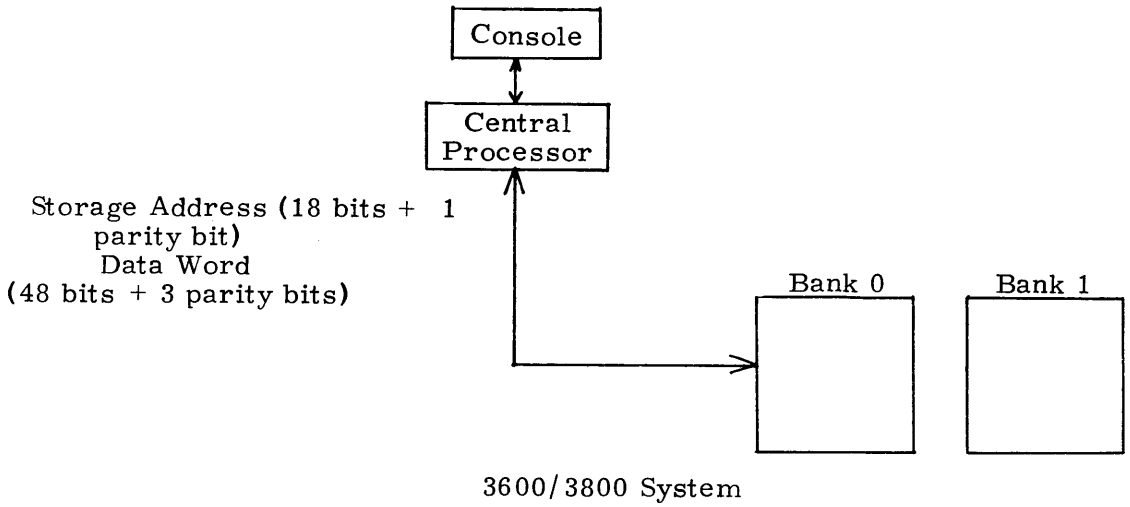


Figure 1-6

The 3803 uses the same logic and principles found in Control Data's 6000 line of computers. With this logic the cycle time for a memory reference is only 800 nanoseconds (. 8 usecs.). This includes both the read and write phases.

The 3400 and 3600 memory modules use 3000 series logic, are built different, and are slower. The cycle time for them is approximately 1. 5 usecs. Figure 1-7 is a picture of the 3600 memory module.

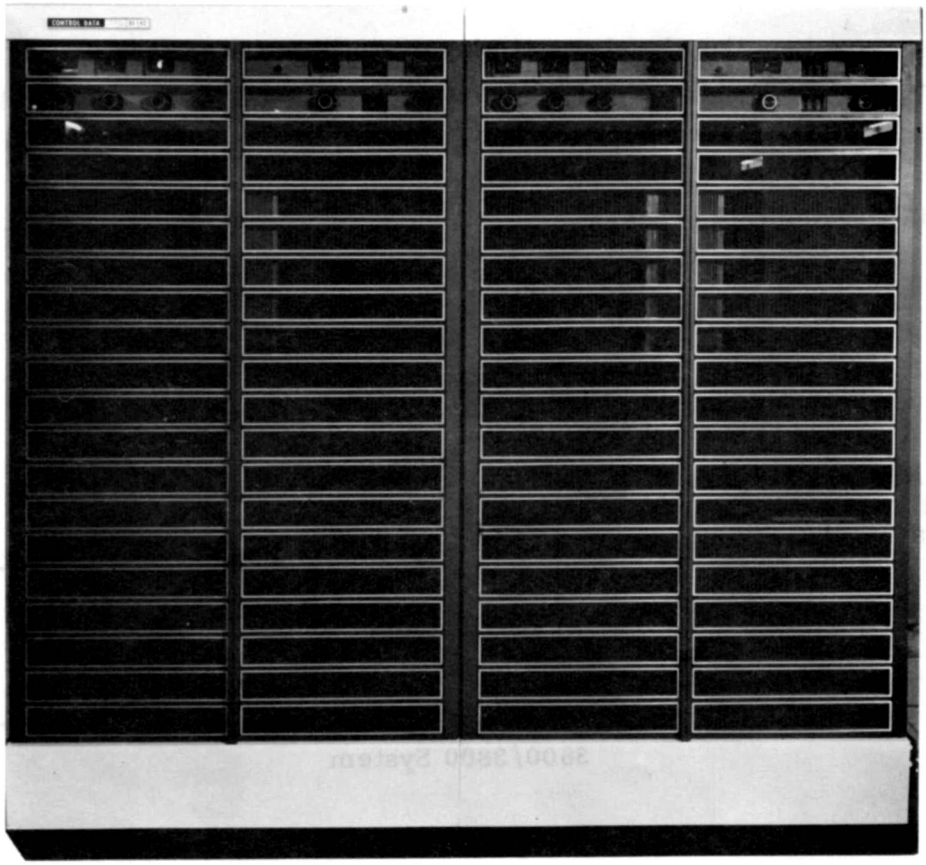


Figure 1-7

The module contains 32K storage addresses. Up to eight of these may be included in a 3600 system. However, in the 3400 system only one bank is allowed.

As long as no parity error occurs the operation is normal. However, three types of parity errors could occur, all of which are monitored on the console.

1. Storage Address Parity Error
2. Instruction Parity Error
3. Operand Parity Error

1. A Storage Address Parity Error occurs when the central processor generates the address, attaches the parity bit, and memory finds an error. Memory in this case will signal the processor to stop. This constitutes an unrecoverable error.

2. An Instruction Parity Error occurs when the central processor reads a 48-bit word as an instruction. If the 51 bits transmitted to it are found to have a parity error, the processor stops. This constitutes an unrecoverable error (3400 it is trapped in interrupt).

3. An Operand Parity Error occurs when the central processor reads a 48-bit word as an operand. If the 51 bits transmitted to it are found to have a parity error, the processor can continue. This constitutes an interruptible condition and can be checked in the program.

INPUT/OUTPUT

Up to this point we have shown the central processor, the console and the memory. From here we would like to show the input/output section of the 34/36/3800 systems. Again, we will concentrate on the 3800 system and any differences in the 3400 and 3600 systems will be noted.

In order to maintain organization and clarity of purpose, we are not at this time going to become deeply involved in the intricacies of the input/output equipment. We will

approach it in the following order:

1. System concept
2. Basic data flow

System Concept

In order to explain the system concept, we are going to show the maximum configuration possible. Maybe no one has a maximum system, but if we understand the overall concept, any smaller system should offer no trouble to us.

Connected to the central processor is the Communications Module (3602/3802). This module does not take up much room; in fact, it only takes up a small portion of the physical module shown in Figure 1-8.

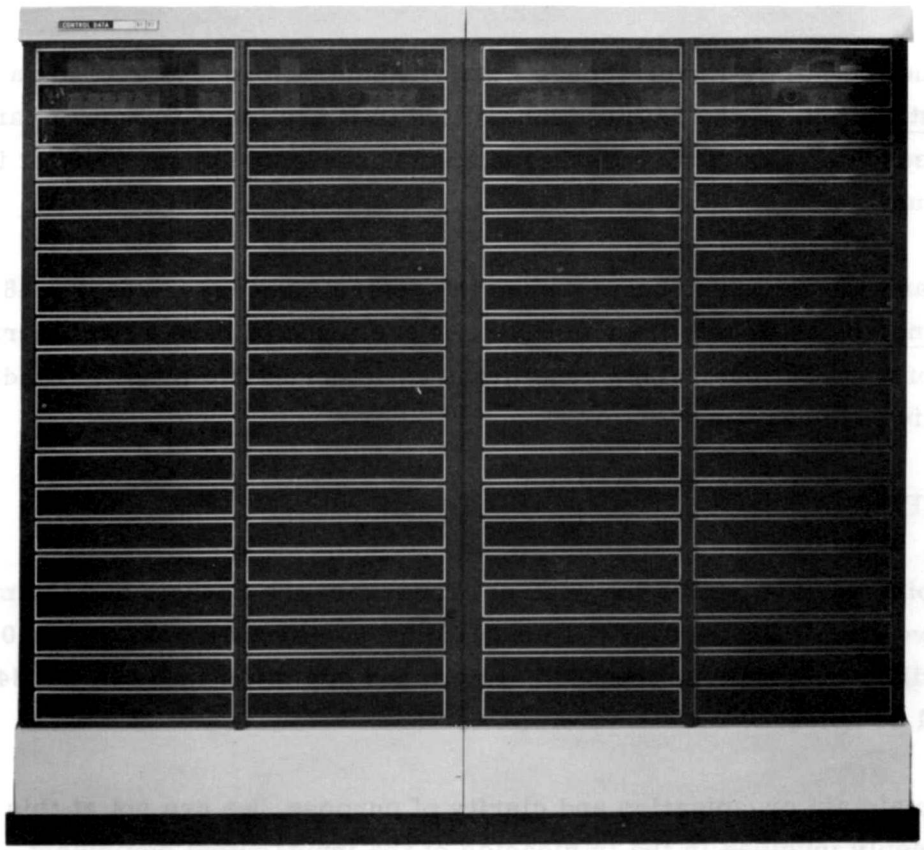


Figure 1-8

If you take a close look at Figure 1-8, the 3602/3802 is on the extreme right hand side. It is only fourteen inches wide. The rest of the physical module contains data channels. There are five data channels included in Figure 1-8. Each has six indicator lights at the top.

The communications module does not do much in itself and at this point it is difficult to explain its exact role in an input/output operation. At this time we might say that it acts as a transfer path between memory and the I/O equipment.

Building our diagram from previous explanations, it now looks like Figure 1-9.

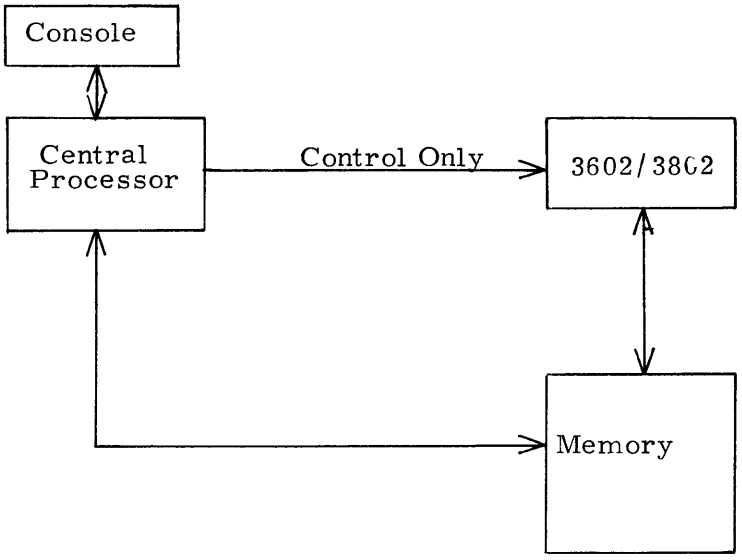


Figure 1-9

Notice how the arrows point in both directions, except between the central processor and the communications module. Control only goes across this line, i.e. never actual data words. If the central processor is to process data from external equipment, the data must first be transmitted to memory and then called from memory and processed.

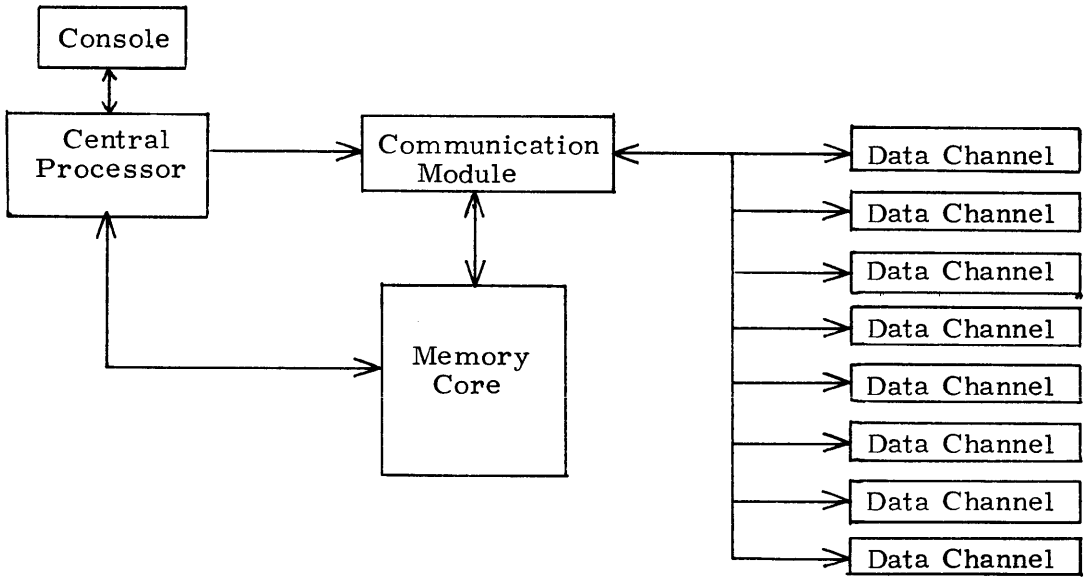
The arrow from the communications module to memory shows that data can be read from memory and transferred to the external equipment or can be read from the external equipment and transferred to memory. Forty-eight bits are transferred across the line along with 3 parity bits for each word. Of course, an 18 bit address is also

transmitted along with the data, whether it be a read or a write.

You might sense that the communications module is much like the central processor with respect to memory. You are right. They both send and receive the same types of signals. Memory couldn't tell the difference. It just reads or writes the requested data at the specified address.

Up to 8 data channels may be physically connected to a communications module. The data channel is a 3406/3606/3806. Each data channel is bi-directional, i.e. information may transfer in either direction.

An expansion of the system would look like Figure 1-10.



Expansion of a 3600/3800 System

Figure 1-10

The data channel, after assuming control from the central processor, directs the flow of information. For an input operation information is assembled from the external equipment. As 48 bits are assembled, the data channel transfers the word to memory via the communications module. For an output operation information is read from memory via the communications module, disassembled in the data channel, and transferred 12 bits at a time to the external equipment.

The cycle time for the 12 bits is 250KC for the 3406/3606 and 1000KC for the 3806. The 3606 and 3806 are interchangeable.

To each data channel a maximum of 8 controllers (synchronizers) may be connected. A controller is a necessary interface between the data channel and the actual unit. Some refer to the controller as the equipment and the actual I/O gear as the unit. We will keep this distinction throughout.

There are many types of controllers that serve as interfaces between data channels and their units. Representative examples are given in the figures below.

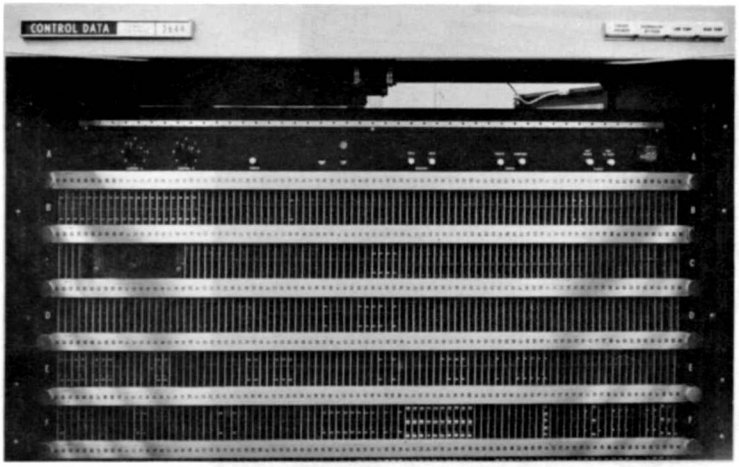
Tape Controller

Tape Unit

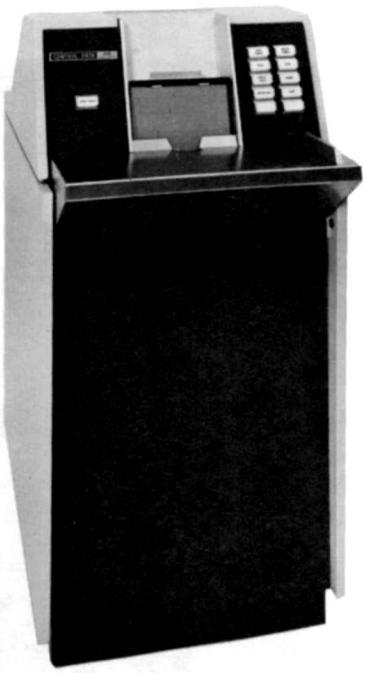
Figure 1-11

Card Reader and Controller

Figure 1-12

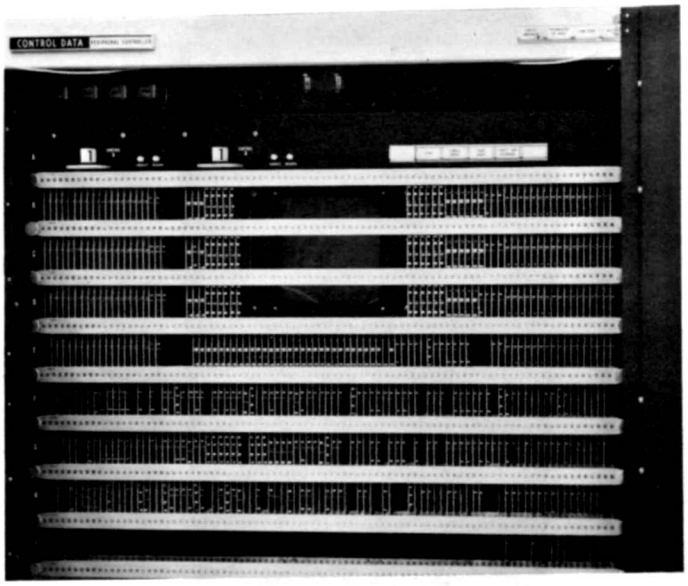


Card Punch Controller



Card Punch

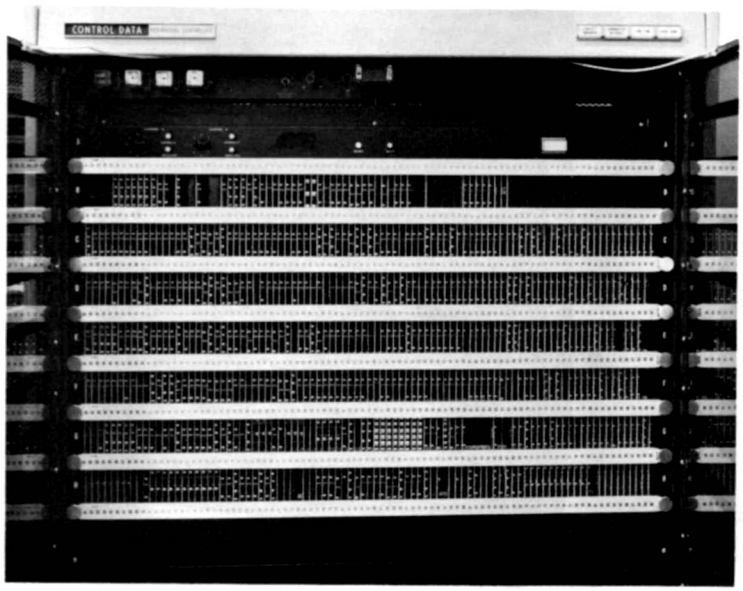
Figure 1-13



Line Printer Controller

Line Printer

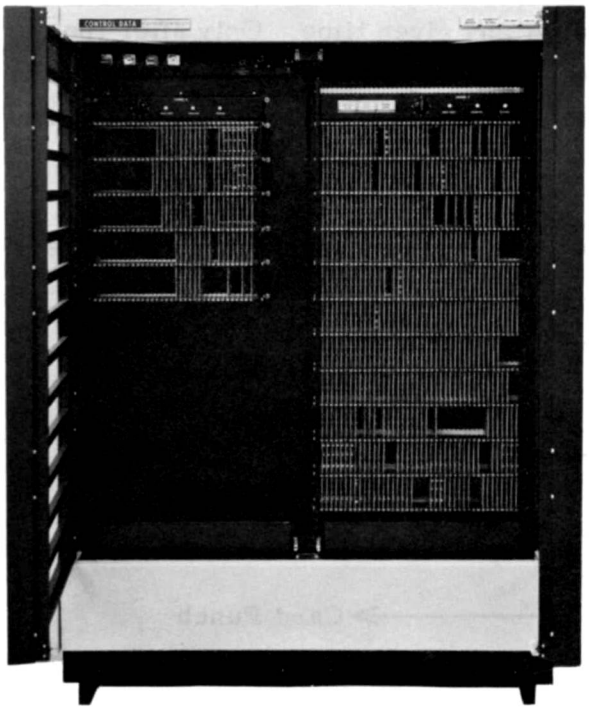
Figure 1-14



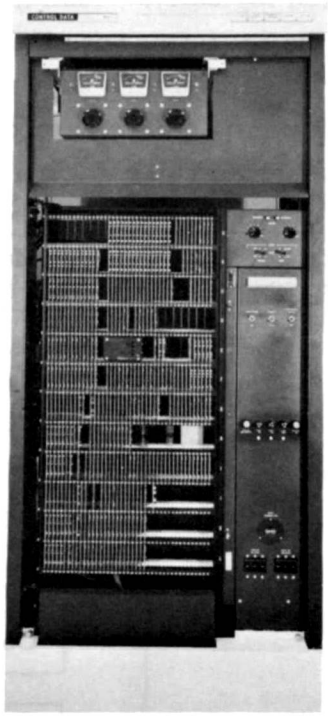
Disk Controller

Disk

Figure 1-15



Drum Controller



Drum

Figure 1-16

Up to 8 controllers may be connected to a data channel physically. However, only one may be connected logically. This means that only one controller and one unit of that controller may be operating with the data channel at any given time. Only after the one operation is finished, may the data channel logically be connected to another unit of the same controller or to another controller and another operation started. An expansion of the system could look like Figure 1-17.

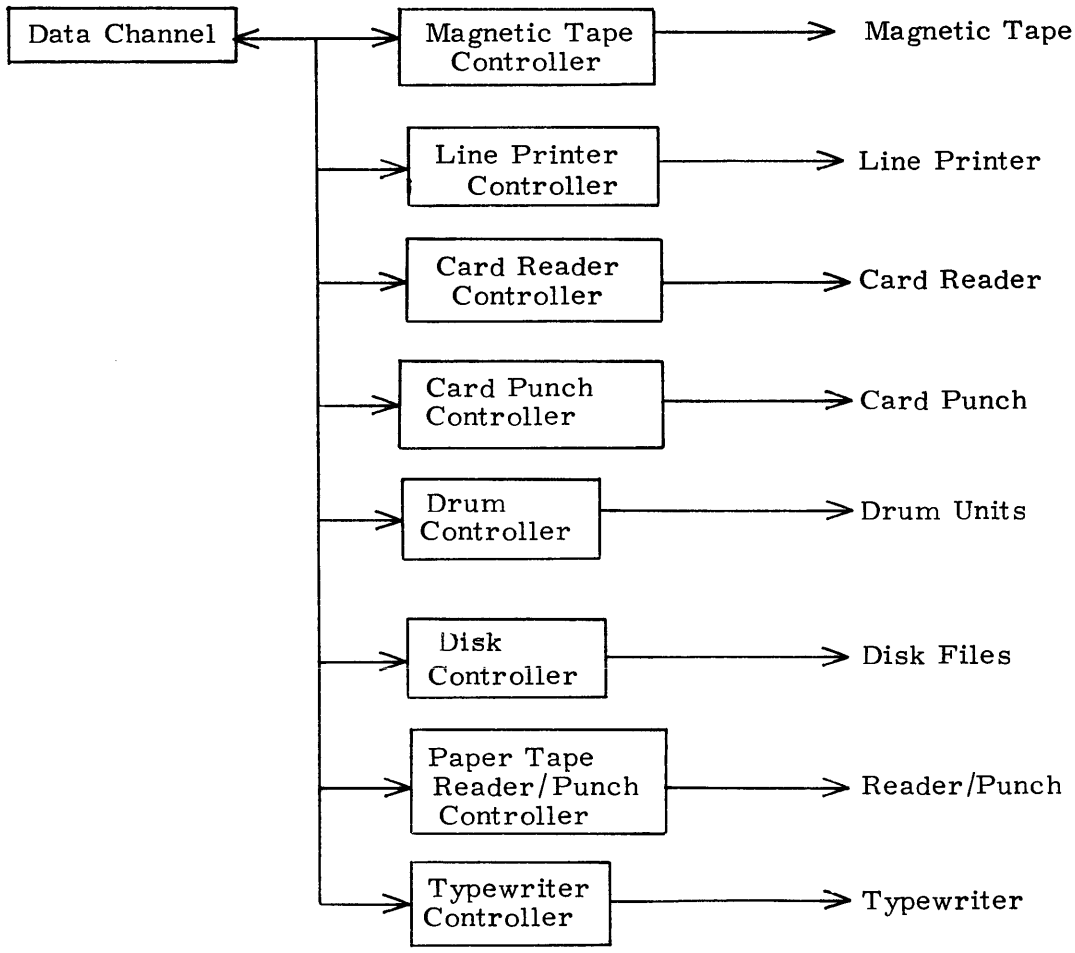


Figure 1-17

To each controller up to 16 units may be physically connected. As of this writing only the magnetic tape controller has this capability. The drum controller can feed up to 8 drums. The disk controller can feed up to 4 disk files. All other controllers can feed only one unit.

The 3400 system's input/output section is virtually the same as the 3600/3800 system except with the limitation that the maximum number of data channels is only 4. This is because the channels are tied directly into memory and each channel services one of memory's access lines. There are no communications modules in the 3400 system.

Basic Data Flow

Data flow for an input/output operation is a transmission of data either from memory to the external equipment (output) or from the external equipment to memory (input). Let's look further into each of these operations.

Write Operation (Output)

During a write operation words are read from memory and transferred to external equipment. Each word read from memory is disassembled as in Figure 1-18.

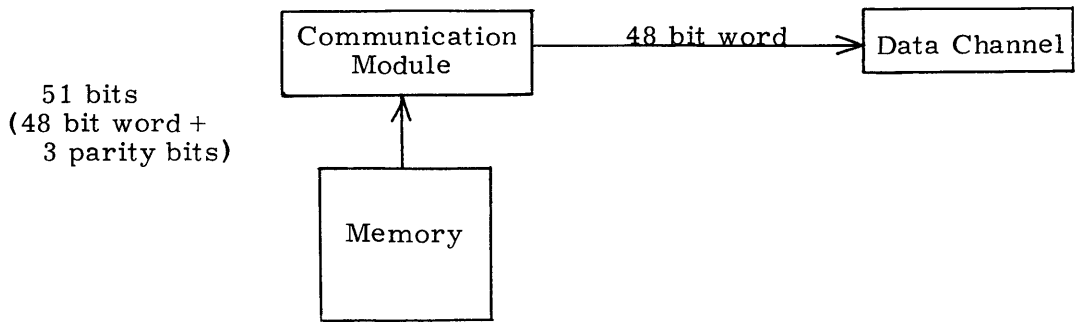


Figure 1-18

Fifty-one bits are transferred from memory to the communications module where the parity bits are checked. If parity is correct, the parity bits are discarded and the 48 bit word is transmitted to the data channel.

The data channel disassembles the word into four 12 bit bytes and transmits them, one at a time, to the presently connected controller as in Figure 1-19.

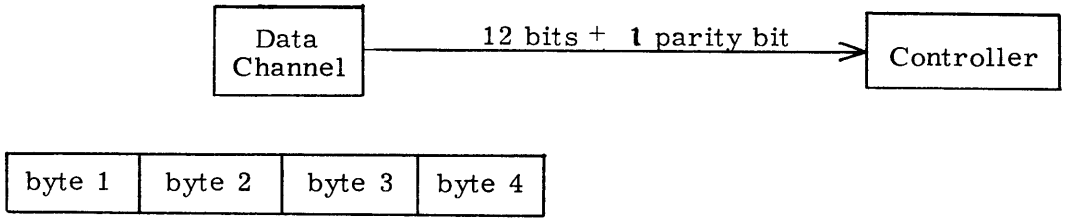


Figure 1-19

As each byte is transferred, a parity bit is generated by the data channel and accompanies it. The controller has a 12 bit parity checker.

As the controller receives each 12 bit byte, it relays the information to the presently connected unit. Controllers vary at this point depending on what type of equipment it services. Let's assume a magnetic tape controller servicing 607 tape units and proceed.

This controller disassembles the 12 bit byte further into two 6 bit bytes. The 6 bit bytes are transmitted, one at a time, to the presently connected 607. The controller generates a parity bit for each 6 bit byte transmitted. The 6 bits plus parity bit comprise the 7 bits per frame recorded on magnetic tape. Figure 1-20 below illustrates this portion of the transmission.

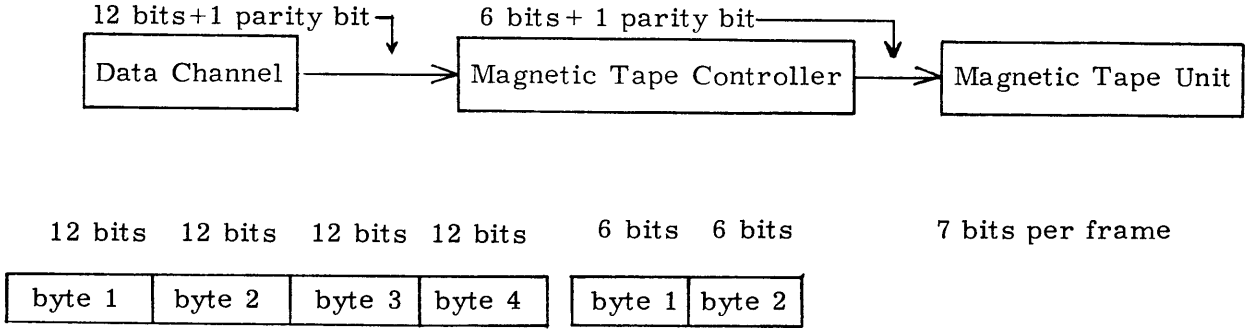


Figure 1-20

How is data transmitted in the opposite direction? Let's take a look.

Read Operation (Input)

During a read operation data is read from external equipment and transferred to memory. Each piece of data is assembled making a 48 bit word before being transmitted by the data channel to memory.

Control is given to the data channel to begin inputing data. The data channel issues a request to the equipment to transmit data. If we again assume magnetic tape controllers and units, data will be read 7 bits per frame.

The frames are read from tape, one at a time. As each frame is read, the parity bit is checked and discarded with only 6 bits of the frame being assembled. When the controller has accumulated two 6 bit bytes, it transmits the 12 bits to the data channel along with a parity bit. The data channel checks the parity bit of the 12 bit byte, discards it, and holds the 12 bits of data. It then requests more data from the controller which in turn requests two more frames to be read. Figure 1-21 is an illustration of the assembly of data.

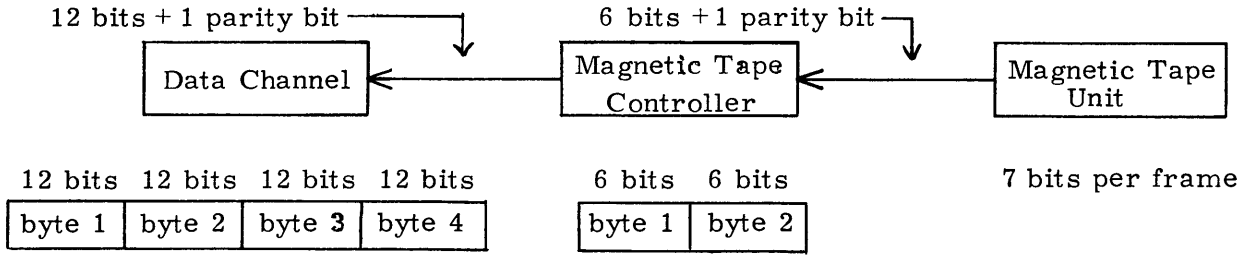


Figure 1-21

Only when the data channel has received 48 bits does it transmit the word to storage. The word is transmitted to storage via the communications module which generates 3 parity bits and stores the 51 bits as shown in Figure 1-22.

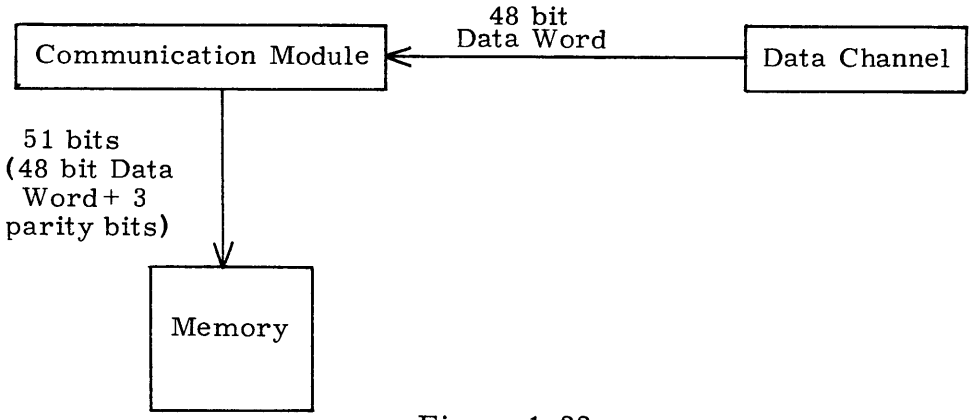


Figure 1-22

This operation continues until the data channel or the central processor terminates it.

SYSTEMS CONFIGURATIONS

At this time we would like to discuss various small systems configurations as an aid to further understanding the functions of modules and how they operate within the system.

Consider the system in Figure 1-23.

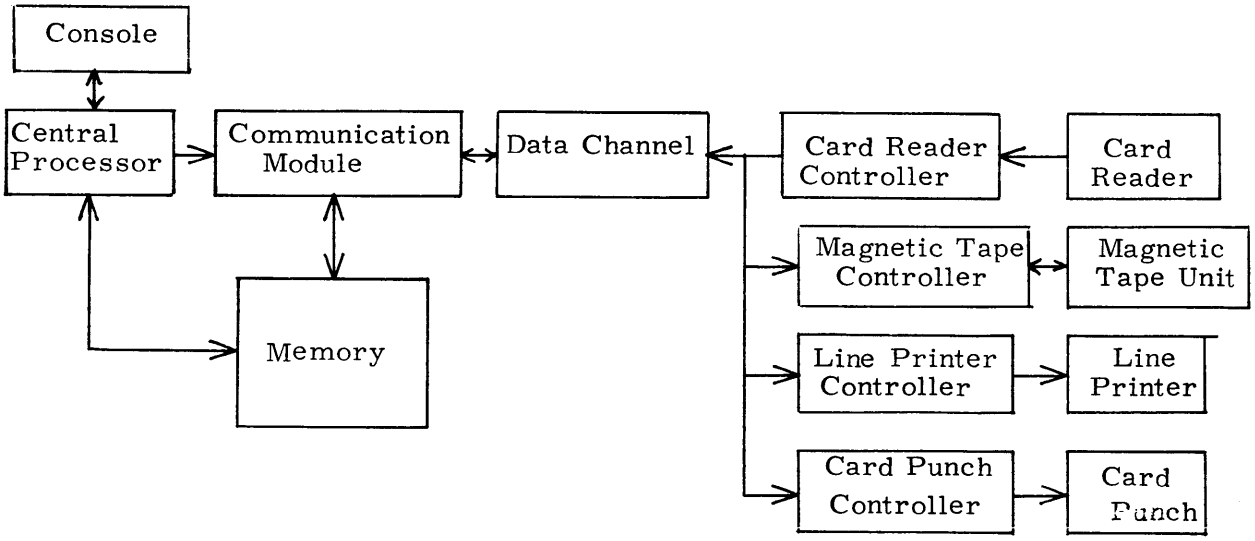


Figure 1-23

This system has one data channel and four types of equipment. Since the data channel is bi-directional, information could be transferred from memory to either magnetic tape, the line printer, or the card punch. Also information could be transferred from magnetic tape or the card reader to memory. Only one operation may take place at any given time, however. This means that maximum efficiency is not attained from this system.

To help alleviate the servicing of I/O equipment only one at a time, consider the system in Figure 1-24.

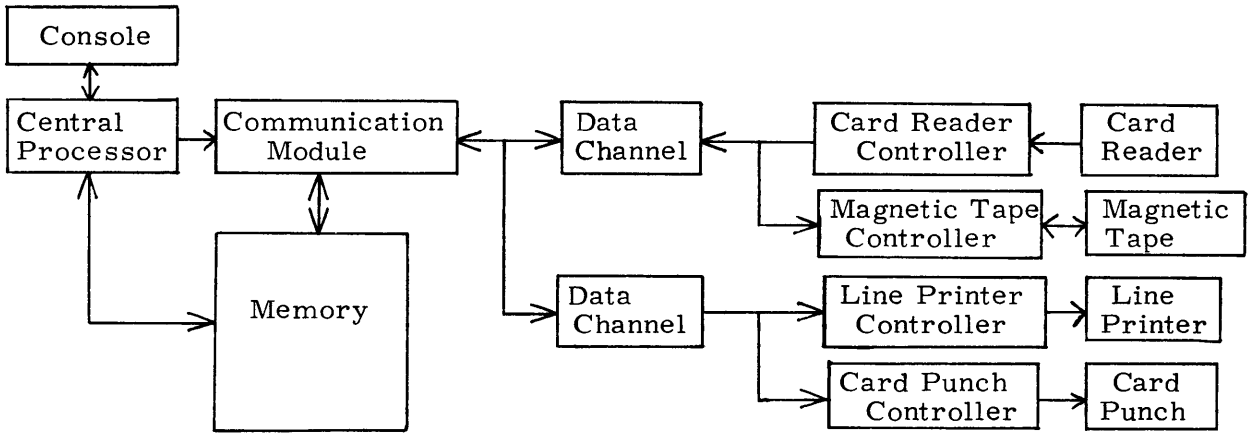


Figure 1-24

This system is more efficient than the previous system. Here, the two data channels may operate simultaneously. For example, the first data channel could be inputing from the card reader at the same time the second data channel is outputing to the line printer. The communications module has a free running scanner to service up to 8 data channels, one memory reference at a time.

MULTI-CHANNEL CONTROLLERS

An important concept used in satellite systems and in systems where it is necessary to operate simultaneously two or more units of the same controller is the concept of the

Multi-channel Controller.

A multi-channel controller is a controller that can be fed by more than one data channel. A bi-channel controller might be diagrammed as in Figure 1-25.

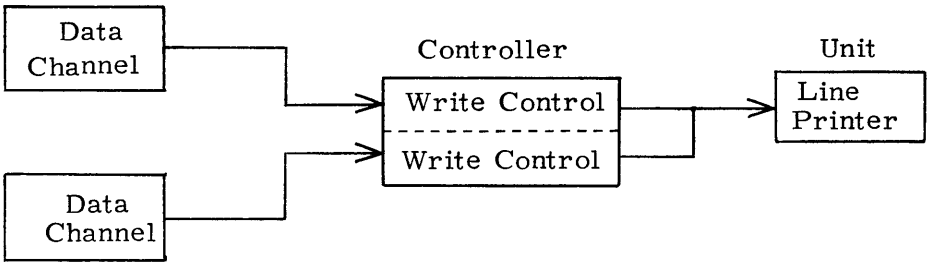


Figure 1-25

An example of a controller of this type is the 3659 (line printer controller).

Each half of the controller has its own separate write controls allowing either channel to transmit information to the line printer. The data channels could be from different computers, thus comprising part of a satellite system.

The largest multi-channel controller may take up to 4 channels. This is the magnetic tape controller which is a 3623 or 3624 (4 x 8 or 4 x 16 respectively). A system might include Figure 1-26.

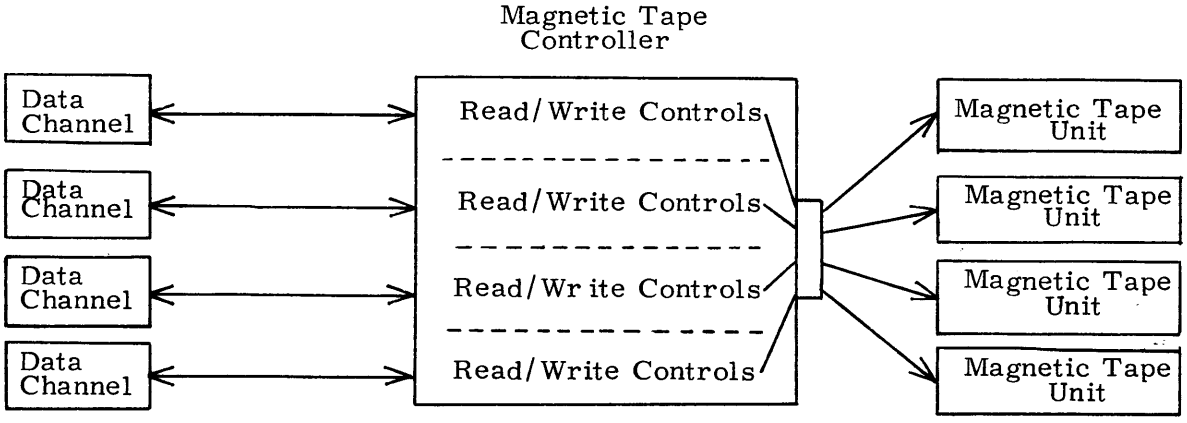


Figure 1-26

The controller has four separate Read/Write controls, each of which are capable of transmitting data to or from the channel and unit simultaneously. This means that the four tape units could be going simultaneously, assuring maximum efficiency.

SATELLITE SYSTEMS

Multi-channel controllers can also be used in satellite systems. Figure 1-27 is an example of a dual 3600 system.

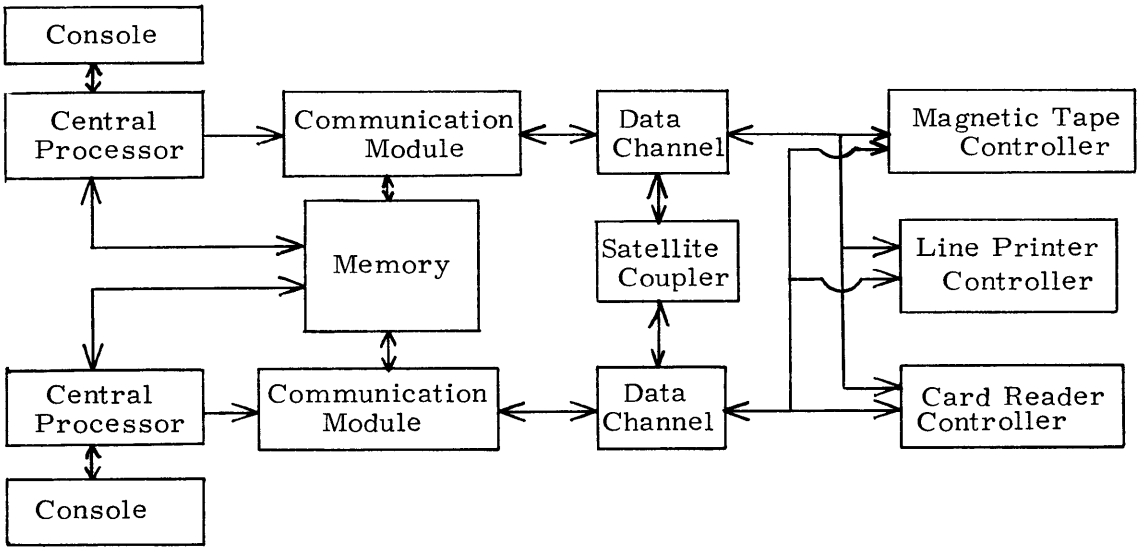


Figure 1-27

In this system the multi-channel controllers are attached to channels of both systems meaning that either processor can reference them. The central processors also share a common memory. Communication between the two processors takes place via the satellite coupler (3682).

SECTION II

THE CENTRAL PROCESSOR

INTRODUCTION

The make-up of the 34/36/3800 central processors varies only slightly. This section will discuss mainly the 36/3800 processors. If there is any significant difference for the 3400 processor, it will be noted.

The central processor is made up of five parts which will be discussed in this section; Arithmetic, Control, Memory (addressing), I/O, and Interrupt.

ARITHMETIC SECTION

The central processor is capable of performing arithmetic operations using the following types of arithmetic:

1. fixed point integer
2. fixed point fractional (except 3400)
3. floating point (3400 option)

The range for fixed point integer is from $-(2^{47} -1)$ through $+ 2^{47} -1$ in increments of 1. The arithmetic is always one's complement.

The range for fixed point fractional is between -1 and + 1 making it very limited. This arithmetic is also one's complement.

The floating point arithmetic entails a combination of fixed point integer and fixed point fractional in that integers, fractions, or mixed numbers can be expressed and operations performed without the programmer adjusting the point. The range for floating point numbers is between -10^{308} and $+ 10^{308}$ which gives programmers plenty of leeway in working scientific problems. This arithmetic is also performed using one's complement.

There are three adders (really subtractors) used to perform the arithmetic; the A adder (48 bit), the Q adder (48 bit), and the U adder (15 bit). A and Q adders feed the A and Q registers with a final result from an add or subtract operation. They always perform operations using one's complement arithmetic. The A and Q adders for some instructions combine to form a 96 bit answer. The U adder is only 15 bits and is normally used for address modification such as the adding of $m + (B^b)$, $y + (B^b)$, or $k + (B^b)$. It usually performs arithmetic using one's complete notation. However, there is an instruction that can set it to two's complement mode, especially if the programmer wishes to obtain 77777_8 from it through normal incrementation of an index register.

The registers in the central processor are all displayed on the console, except for the S register which is non-program addressable. The registers discussed in the arithmetic section are A, Q, index registers, and D.

The A register is called the Main Arithmetic Register and looks like Figure 2-1.

Figure 2-1

The A register is 48 bits. It contains nearly all results from arithmetic and logical operations. The A adder feeds this register directly.

Sometimes the A register works in conjunction with the Q register to form a 96 bit result. This is done, for example, when forming the product from a multiplication operation.

Quantities in the A register can also be shifted; either alone or in conjunction with Q. Quantities are shifted by binary bit positions either to the left (normally end-around) or to the right (end-off).

The Q register is called the Auxiliary Arithmetic Register and looks like Figure 2-2.

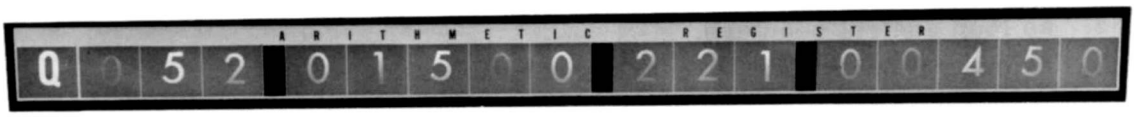


Figure 2-2

The Q register is 48 bits. Very often it is used in conjunction with the A register to form a 96 bit quantity from arithmetic operations. The Q adder feeds this register directly.

Sometimes this register represents a mask for logical and search operations. Quantities compared or searched can be masked with the quantity in Q before the comparison takes place.

Quantities in the Q register can also be shifted; either alone or in conjunction with A. Quantities are shifted by binary bit positions either to the left (normally end-around) or to the right (end-off).

There are six index registers that are shown through one octal display. Typical examples are shown in Figure 2-3.

Figure 2-3

Index registers 1 and 2 are shown. The others could be shown merely by changing a switch on the console.

Each index register is 15 bits. The registers perform the following functions:

1. as address modifiers for relative addressing.
2. as loop control counters.
3. as control quantities for search operations.

As address modifiers the contents of an index register serves as a "pointer" relative to a base address when a memory table is processed.

As loop control counters a loop can be repeated the exact number of times and then discontinued. Each time a pass is made within a loop a check can be made for possible termination of the loop.

As control quantities for search operations the contents of an index register can determine the number of values in a table to compare. Also, it can determine which ones to skip.

The D Register (also referred to as the "Flag" Register) is 48 bits and looks like Figure 2-4.

Figure 2-4

This register has a two-fold purpose.

1. as temporary storage.
2. as a flag holder (hence "Flag" register).

As temporary storage this register affords the programmer a relatively fast and efficient method for holding a value temporarily before being processed. There is a register-to-register instruction that will transmit any register to D and return it when needed. This is faster than storing the value in memory and then retrieving it.

As a flag holder the D register has 48 bit positions, each position representing a flag that is set or clear. There is a flag test instruction that will test any flag to determine its status. Also, of course, there is an instruction that can initially set or change any one flag.

CONTROL SECTION

The control section of the central processor directs the operations required to execute instructions and establishes the timing relationships needed to perform these operations in the proper sequence. It acquires an instruction from storage, interprets it, and sends commands to other sections.

Before the control section of the central processor transmits an address to memory for the next instruction or instruction set, it merges two registers together; the P Register and Instruction Bank Register. This forms an 18 bit address which is sent to memory. A discussion of the P and Instruction Bank Register follows. (In the 3400 only the P register exists and only 15 bits are transferred since it is a one bank machine.)

The P Register is called the Program Address Counter, is 15 bits, and looks like Figure 2-5.

Figure 2-5

The P register holds the address of the current instruction or instruction set. It is controlled by a two's complement counter, so that, as the contents of P are incremented it counts from 00000 through 77777_8 and then reverts back to 00000.

The contents of P can be changed accordingly:

1. (P) + 1
2. (P) + 2
3. changed by a "jump address"

The contents of P are advanced by one when a "full exit" is taken. This means that the central processor has finished with the present instruction or instruction set and is ready for the next.

The contents of P are advanced by two when a "skip exit" is taken. This happens on a

jump instruction where a test is made and following the test a full exit or skip exit is taken.

The contents of P are changed by a "jump address" when a jump instruction is encount ered and the jump condition is satisfied. Program control will reference a new address usually out of sequence with the main program and this new address will replace the contents of P.

The P register does not comprise the complete address that is transmitted to memory when referencing instructions. As alluded to before, an 18 bit address is always trans mitted to memory from the central processor. The contents of P is always merged with the Instruction Bank register.

The Instruction Bank Register (IB) is a 3 bit register and is shown in Figure 2-6 (non-existent in 3400 since it is a one bank machine).

Figure 2-6

The Instruction Bank register determines which bank is to be referenced for instructions. A maximum of eight banks may be referenced; hence the range of 0-7 for IB. It is vitally important that the programmer be aware of the bank registers and know how to change them if he uses a multi bank system (36/3800). He may have the correct 15 bit address, but if he references the wrong bank, he will receive incorrect information.

Program control, in reading instructions, then, forms an 18 bit address which is really IB-P and forms it as in Figure 2-7.

Figure 2-7

A special note might be added on the concept of incrementing the program counter in a multi -bank system. It is important to realize that only the P register is incremented as the program advances (not jumps). When the counter gets to address 77777_8, it reverts to 00000 but within the same bank. Program control will not jump banks. The only way to transfer program control to another bank is by use of a Bank Jump instruction. This will then actually change IB to the new bank.

The U Register is called the Program Control Register, is 48 bits, and looks like Figure 2-8.

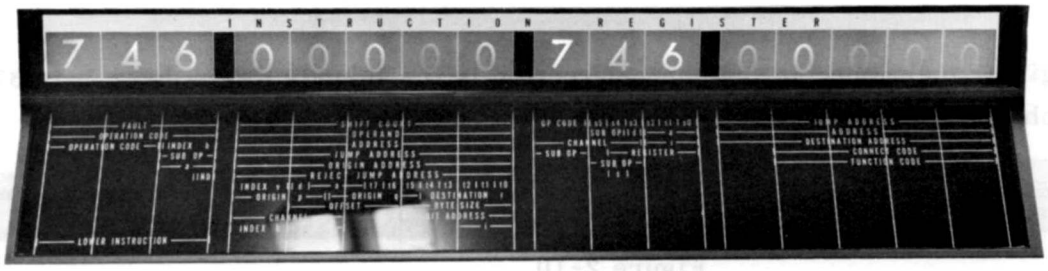


Figure 2-8

When program control reads a word from memory for execution, the word enters this register. This word may represent either two 24 bit instructions or one 48 bit instruction. If it represents two 24 bit instructions, the upper one is executed first. Program control then half exits (no memory reference) and executes the lower one (assuming no jumps). If the word represents one 48 bit instruction, a full exit takes place after execution. The full exit causes program control to advance the P register and reference the next word and the process is repeated.

When an instruction is executed, it may direct control to read or write an operand (data) from memory. The instruction forms a 15 bit address to transmit. In the 3400 it then would transmit it to memory. However, in the 36/3800 systems, since an 18 bit address must be transferred, a 3 bit Operand Bank Register is joined before transmission.

The Operand Bank Register (OB) is 3 bits and is shown in Figure 2-9.

Figure 2-9

The Operand Bank Register determines which bank is to be referenced for operands. A maximum of eight banks may be referenced; hence the range of 0-7 for OB. The programmer can set this bank register to any of these values. When he does, all succeeding references to memory for operands will go to this bank. This bank will continue to be referenced for operands until explicitly changed by the programmer.

The last register mentioned in the control section is the Bounds Register. It is 37 bits and is shown in Figure 2-10.

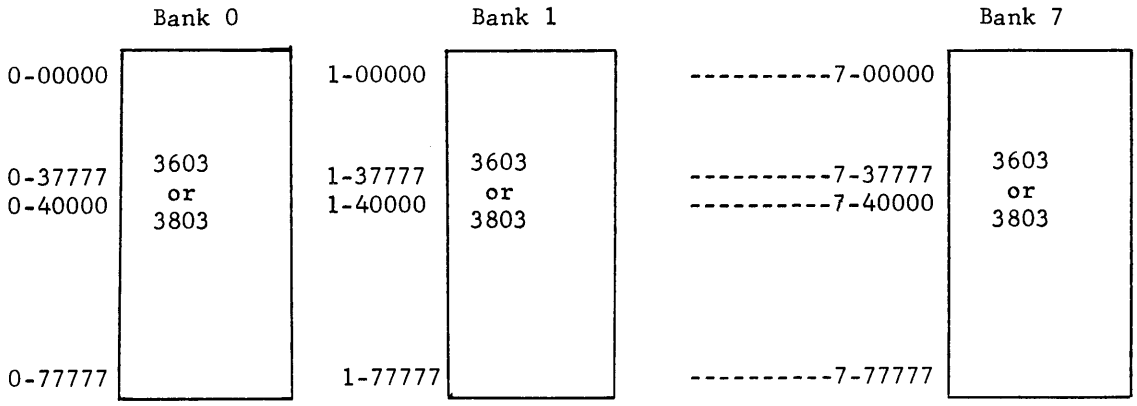
Figure 2-10

This register regulates the Bounds principle- an interruptible feature. Thirty-six of the bits represent two 18 bit addresses, within which program control must operate. The two addresses are an upper and lower bounds. If program control attempts to jump or write out of bounds and the interrupt system is active, the attempt will be blocked and interrupt will occur. Even with the Bounds principle set up operands may still be read out of bounds (except in 1604 mode) with no effect.

The 37th bit (highest order bit), if set, will remove the upper bounds. The lower bounds would still remain in effect.

MEMORY SECTION

The memory exists external to the central processor. It is composed of storage addresses of up to 262, 144 (32, 768 for the 3400) words. Figure 2-11 illustrates the eight bank maximum.



Maximum 36/3800 Memory

Figure 2-11

The uppermost three bits of the 18 bit storage address transmitted determines which bank is referenced. The lower fifteen bits determine the address within that particular bank.

When an address is referenced, 51 bits are read or stored at that address. The upper three bits of the memory word are parity bits. A "read" causes the word to be read and transmitted to the central processor or to some I/O device as we shall see later. The word still remains at the address. A "write" causes a new word to be written at the address. If a word is written, the 3 parity bits are generated by the central processor and 51 bits are transmitted to memory.

Figure 2-12 shows the format for the memory word.

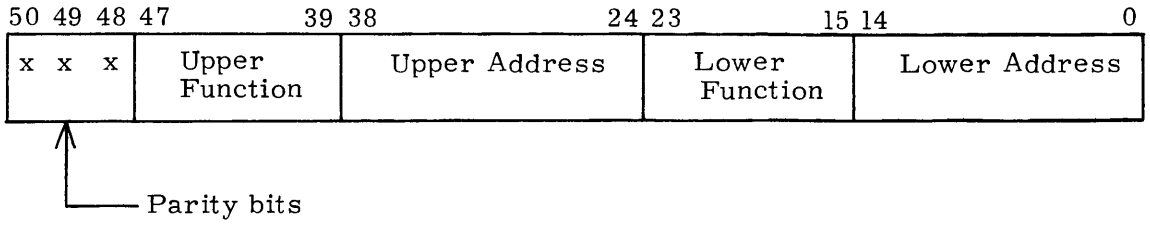


Figure 2-12

The word is formatted according to two 24 bit instructions even though the word may be an instruction or an operand. The parity bits follow this pattern.

1. parity bit 48 is associated with bits 0-14 (lower address)
2. parity bit 49 is associated with bits 24-38 (upper address)
3. parity bit 50 is associated with the sum of bits 15-23 and 39-47 (functions)

Parity is always odd. The lower address and upper address portions also represent the execution portions of 24 bit instructions. These may be changed in memory using the partial read and partial write instructions available in the 34/36/3800 repertoire of instructions.

I/O SECTION

At this time we make no attempt to explain every detail about how the I/O section of the system performs. This is done later. However, at this time it is appropriate to discuss some of the details as a follow-up to the general system explained in Section I.

To each data channel a maximum of eight controllers may be physically connected. To each controller a maximum of eight controllers may be physically connected (magnetic tape). However, only one controller and one unit may be logically connected to a data channel at any given time. This means that only one controller and one unit of the controller may be operating with a data channel. If there is need to operate another unit of the controller with the same data channel or another controller and unit with the same data channel, the second operation must wait until the first is completely finished (channel becomes not busy) before it can be initiated.

Generally, how does a programmer connect anything to a data channel and initiate activity? In order to explain this let us set up a system.

Suppose the system has four data channels. The programmer refers to these channels by number. The range of these channels is 0-3.

Next, assume channel 0 feeds two controllers - a line printer and magnetic tape; channel 1 feeds two controllers - a card reader and magnetic tape; channel 2 feeds two controllers - a line printer and card punch; channel 3 feeds a drum. The system is as in Figure 2-13.

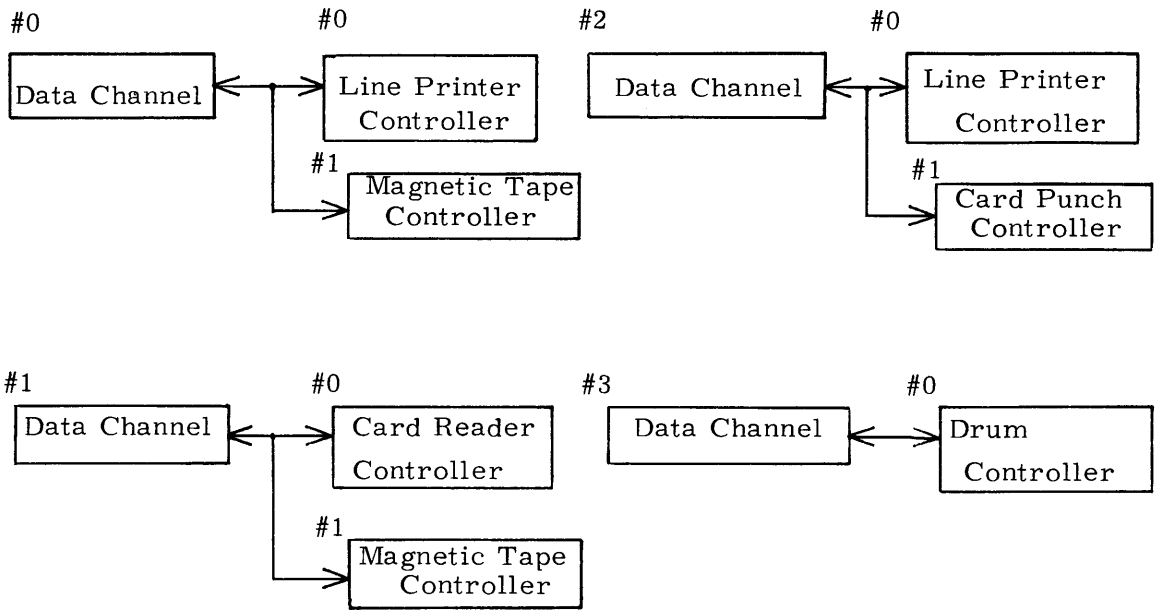


Figure 2-13

A data channel is specified by use of its number which is hardware built in. A controller is also specified by use of its number. At the entrance to each controller there is a dial or switch having numbers ranging from 0-7. Each number represents one of the eight maximum controllers physically connected to the data channel. The programmer references the controller by its designated number. All controllers to one data channel must have different numbers.

A programmer must know beforehand how these numbers are set up. If he thinks controller #1 is the card reader when it really is the line printer, and he tries to read from it. he is going to run into trouble.

According to the diagram, the programmer might connect to data channel #0, equipment #0. He has effectively connected the line printer. This is done by program control through the use of a CONNECT instruction. The CONNECT instruction also allows a unit number to be specified. However, since only one line printer can be physically

attached to its controller, this number is ignored.

If later in the program the programmer connects to data channel #0, equipment #1, the magnetic tape controller will be connected. This automatically removes the connection to the line printer. The CONNECT instruction again allows for a unit number. Here the unit number is interpreted since more than one unit may be physically attached to the magnetic tape controller. The unit is the tape handler or tape drive. It also has a dial of numbers ranging from 0-7. The programmer indicates which tape drive off the tape controller he wants connected. He does this also by specifying its number. Figure 2-14 shows a further breakdown of channel #0.

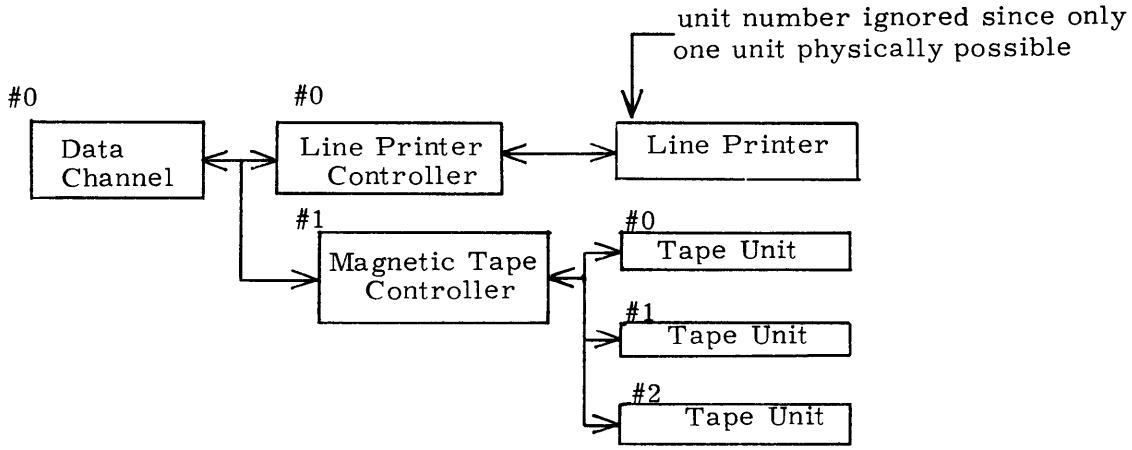


Figure 2-14

A programmer might connect to channel #0, equipment #1 and unit #0. The tape unit is connected. If later he wishes to use tape unit #2 he first connects with another CONNECT instruction. The same data channel and controller numbers are specified but with the new unit. He will have lost his old tape connection. From this it is important to keep in mind that, logically, only one equipment and one unit may be connected to a data channel at any time.

With respect to our initial diagram what device would have been connected with:

1. channel #1, equipment #1 ?
2. channel #2, equipment #0 ?
3. channel #3, equipment #0 ?

If you picked magnetic tape, line printer, and drum, respectively, you are right.

Once a unit is connected to a channel, there are function codes that should be sent prior to reading or writing. On a tape unit, for example, the density and format should be set This is done by the use of an external function instruction (EXTF). Here the programmer specifies only the data channel and issues the proper code. The code will automatically go to the unit last connected to the channel. The codes for each equipment can be obtained in its reference manual. More will be explained about the types of function codes in Volume II.

After functions are set up, actual read or write operations may begin. A Read operation is interpreted to mean a transmission of data from the external equipment to memory. A Write operation is interpreted to mean a transmission of data from memory to the external equipment. There is a READ or WRITE instruction that will do either.

These instructions are very similar. They both specify a Control Word Address, an address in memory which must be preset with a control word. The control word determines where in core the information is to be read or written and how large a block. The control word also contains special features such as chaining, terminating upon end-of-record, scatter reads, and scatter writes. More on this will be explained in Volume II.

When the data channel is busy, either reading or writing, another operation cannot be initiated. However, status checks can be taken of the operation at any time. The programmer does this by use of the COPY instruction. Status could include any of the following.

1. Control Word Address
2. Control Word
3. Channel and Equipment Status

The first two involve what is mentioned above. The channel status indicates whether a parity error occurred during transmission of data. It also indicates if the channel is

reading or writing. The equipment status indicates such things as Ready, Busy, Reserved, End of Operation, Interrupt, etc. Some of the status definitions will be grouped and explained in Volume II.

INTERRUPT SECTION

Since the 3400 and 36/3800 systems differ so greatly, the 36/3800 will be explained first. Then the 3400 differences will be mentioned.

Interrupt in the 36/3800 systems is based on priority. There are 48 sources of interrupt and they are in the following order of priority.

1. Shift Fault - lowest
2. Divide Fault
3. Exponent Overflow Fault
4. Exponent Underflow Fault
5. Arithmetic Overflow Fault
6. Interrupt #1
7. Interrupt #2
8. Internal Reject
9. Real Time Clock
10. Storage Reference Fault
11. 1604 Mode
12. Trace Mode
13. Bounds Fault
14. Illegal Instruction Fault
15. Operand Parity Error
16. Manual Interrupt
17. Data Channel 00
18. Data Channel 01
19. Data Channel 02
20. Data Channel 03
21. Data Channel 04
22. Data Channel 05
23. Data Channel 06
24. Data Channel 07
25. Data Channel 08
26. Data Channel 09
27. Data Channel 10
28. Data Channel 11
29. Data Channel 12
30. Data Channel 13
31. Data Channel 14
32. Data Channel 15
33. Data Channel 16
34. Data Channel 17
35. Data Channel 18
36. Data Channel 19
37. Data Channel 20
38. Data Channel 21
39. Data Channel 22
40. Data Channel 23
41. Data Channel 24
42. Data Channel 25
43. Data Channel 26
44. Data Channel 27
45. Data Channel 28
46. Data Channel 29
47. Data Channel 30
48. Data Channel 31 - highest

Shift Fault

A Shift Fault interrupt condition occurs only upon the execution of one of the six shift instructions. It happens when the instruction attempts to overshift a register. An overshift is a shift greater than 48 for the ARS, QRS, ALS, and QLS instructions and greater than 96 for the LRS and LLS instructions. If an attempt is made, the shift is blocked and the interrupt condition occurs.

Divide Fault

A Divide Fault interrupt condition can occur only on a divide instruction. The DVI, DVF, and FDV instructions are the only ones. The fault occurs if the divisor is zero or if the quotient exceeds the capacity of the answer register.

Exponent Overflow Fault

This fault occurs only on a floating point instruction. It occurs when the resultant exponent exceeds the modules of its format, i. e. greater than 2^{1777_8}.

Exponent Underflow Fault

This fault also occurs only on a floating point instruction. It occurs when the resultant exponent exceeds the modulus of its format, i. e. becomes less than 2^{-1777_8}.

Arithmetic Overflow Fault

This fault occurs when the result of an add or subtract operation exceeds the modulus of the register, i. e. when the absolute value becomes greater than $2^{47}-1$.

Interrupt #1

A direct interrupt from another computer or device can be tied in and sensed.

Interrupt #2

A direct interrupt from another computer or device can be tied in and sensed.

Internal Reject

The Internal Reject interrupt occurs when this computer attempts I/O and the equipment is not physically attached to the system.

Real Time Clock

This condition results when the real time clock reaches a preset value.

Storage Reference Fault

This fault occurs when the central processor tries to reference a non-existent bank of core. Through this feature a programmer can determine how much core is available or which core is down (inoperative).

1604 Mode

This interrupt is caused when a 1604-3600 incompatible instruction is ready for execution. This interrupt allows the 3600 to interpret 1604 instructions and convert them to 3600 logic.

Trace Mode

This interrupt is caused by a jump instruction in which the jump condition is met. The interrupt will take place before the jump takes place. This feature allows the programmer to "trap" jumps.

Bounds Fault

This fault pertains to the Bounds Register which contains two 18 bit addresses within which the program is to operate. If the program attempts to jump or write out of bounds, the interrupt takes. This is effectively a memory protect feature.

Illegal Instruction Fault

This instruction prevents the use of illegal parameters in certain instructions.

Operand Parity Error

This interrupt occurs if an operand (data) read from memory contains a parity error.

Manual Interrupt

This interrupt is caused by depressing the manual interrupt switch on the console.

Data Channel

Each data channel is capable of interrupting the central processor. A channel may interrupt the central processor for;

a. an end-of-operation
b. an error (storage error, parity error, etc.)

The interrupt system is initially set up by making it active. This can be done by executing the INTERNAL FUNCTION instruction with the proper code. Without the interrupt system active no interrupt will take place. All would be ignored.

Besides the interrupt system being active, another condition must be present in order for an interrupt condition to be recognized. The mask bit for the condition must be set. The programmer can do this in his program.

The Interrupt Mask Register is a 48 bit register and is shown in Figure 2-15.

Figure 2-15

Each bit position in the register represents one of the 48 possible interrupt conditions. The uppermost bit represents Data Channel 31. The lowermost bit represents

Shift Fault. A bit set means that the condition associated with that bit will be recognized as an interruptible condition if and when it occurs. If the bit is not set, an interrupt caused by that condition will not force the program into interrupt. It would be ignored. In this way the programmer can choose which conditions he wants to monitor and which ones he doesn't.

Not all the bits in the Interrupt Mask register can the programmer control. You may notice from the picture of the register that the leftmost 32 bits representing the data channel interrupts are all 1's. These interrupts are called the External Interrupts and are non-programmable - they cannot be changed. These are always automatically monitored. The 16 to the right, however, are programmable and may be changed. These are called the Internal Interrupts and the programmer can choose to monitor or ignore these types of interrupts. These bits are usually set in the register at the entrance to a routine within a program or at the entrance to the program itself.

Along with the Interrupt Mask register there exists a 48 bit Interrupt Register as shown in Figure 2-16.

Figure 2-16

The register works in conjunction with the Interrupt Mask register. Here also, each bit represents one of the possible interrupts. A bit in this register is set when an interruptible condition monitored actually occurs.

The bit causing the interrupt is cleared by the programmer by means of the INTERNAL FUNCTION instruction, if it is an internal interrupt. If it is an external interrupt (data channel), either a CLEAR CHANNEL or EXTERNAL FUNCTION instruction will drop the interrupt.

When interrupt occurs, program control goes to bank 0 address 00001. A return instruction is placed by the hardware at address 00000. The interrupt program stores registers and determines the source of interrupt, clears it as mentioned above, and processes it. When he jumps to address 00000 the main program will be resumed.

The 3400 interrupt systems differ in that there are three categories of interrupt. Category I is the largest and consists of the following:

1. Shift Fault
2. Divide Fault
3. Arithmetic Overflow Fault
4. Exponent Overflow Fault
5. Exponent Underflow Fault
6. Channel 0 Becoming Inactive
7. Channel 1 Becoming Inactive
8. Channel 2 Becoming Inactive
9. Channel 3 Becoming Inactive
10. Channel 0 I/O Transmission Parity Error
11. Channel 1 I/O Transmission Parity Error
12. Channel 2 I/O Transmission Parity Error
13. Channel 3 I/O Transmission Parity Error
14. Channel 0 External Interrupt
15. Channel 1 External Interrupt
16. Channel 2 External Interrupt
17. Channel 3 External Interrupt
18. Out Of Bounds
19. Manual Interrupt
20. Time Interrupt
21. Operand Parity Error
22. Instruction Parity Error

Program control transfers to address 00007 for this type.

Category II interrupts occur when a floating point instruction is attempted when the floating point option is not present. Program control transfers to address $30\text{-}33_8$ depending on the function code of the instruction attempted.

Category III interrupts occur when an illegal instruction is attempted. Program control transfers to address 20_8.

SUMMARY

This section has dealt with the general, but with a little more detail than Section I, concepts of the central processor. We have shown the basic parts of the central processor, parts that will help the programmer to understand what is going on when he programs the machine.

The input/output section has been explained in more detail than in Section I of this volume so as to prepare for the actual programming of it in Section III of this volume and in Volume II.

The interrupt section also was introduced because a systems programmer would have need for this in programming the I/O equipment as well as in the programming of the macros.

SECTION III

PROGRAMMING THE 34/36/3800 SYSTEMS

INTRODUCTION

The objective of this section is to present to the reader a step-by-step process of learning the definition and use of some of the computer instructions through problem solving. We are given a problem, we discuss the instructions needed to solve the problem, and then we present the solution to the problem.

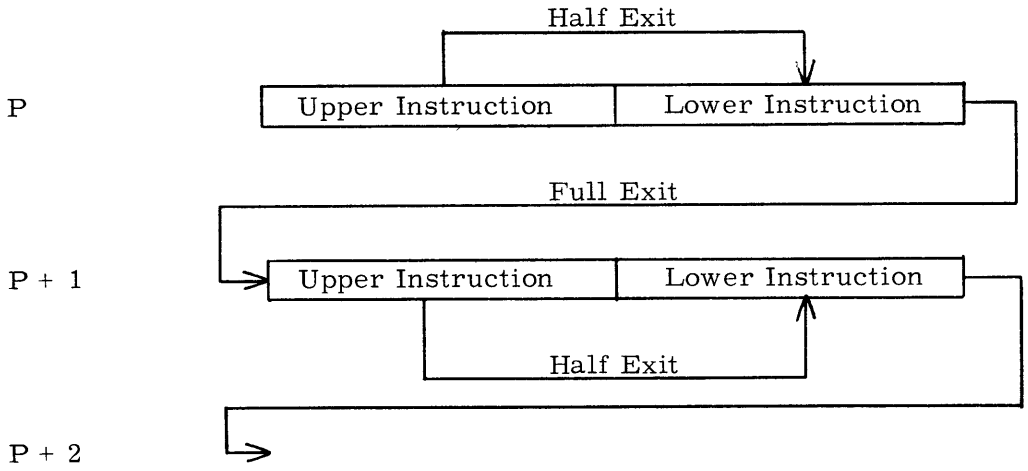
Before we can solve any problems, we must understand instruction format.

HARDWARE INSTRUCTION FORMAT

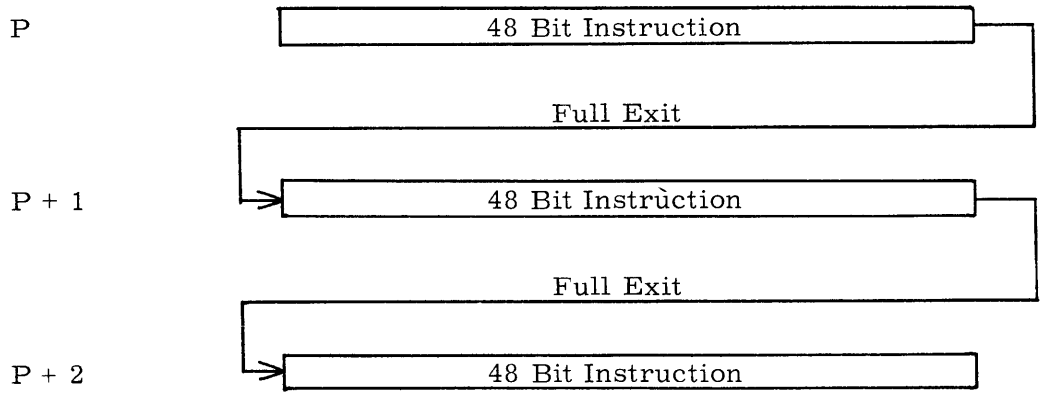
Instructions are divided into two categories.

1. 24 bit instructions
2. 48 bit instructions

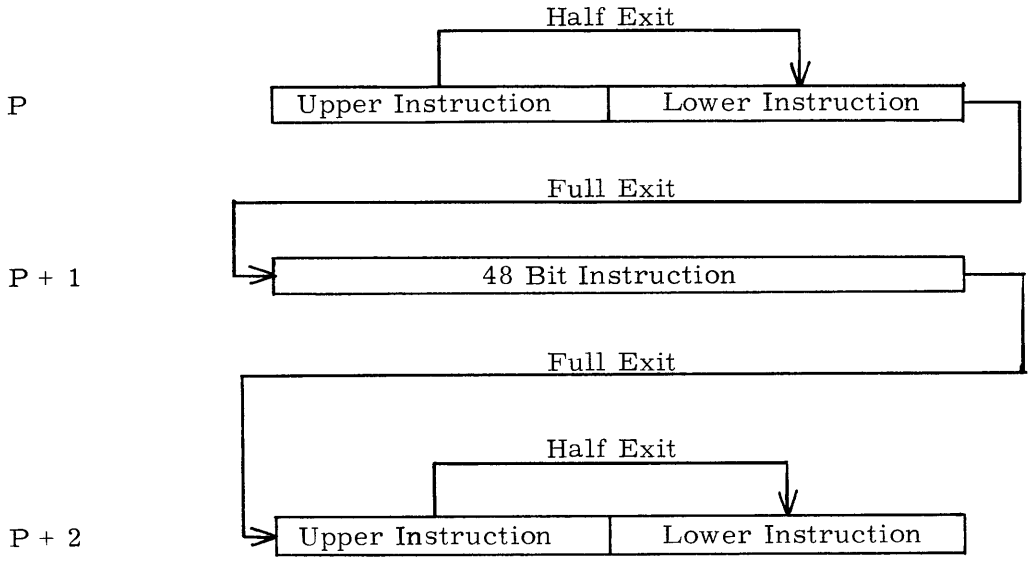
A memory word read to be executed consists of either two 24-bit instructions or one 48-bit instruction. If it consists of two 24-bit instructions, the one on the left (upper instruction) is executed first. When the upper instruction is executed a "half exit" takes place (assume no jump instructions) upon which the instruction to the right is executed (lower instruction). After the lower instruction is executed a "full exit" takes place which advances the Program Address Counter by one and references memory for a new 48-bit word.



If the next word consists of one 48-bit instruction, it will be completely executed. After execution, a full exit is taken to read the next word (never a half exit).

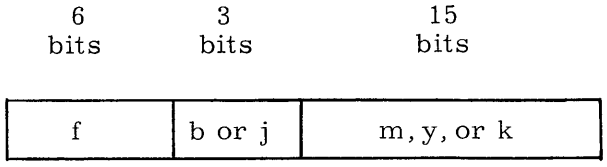


The two categories of instructions may be mixed.



A 48 bit instruction must start at the left side of the word, i.e., it cannot start at the middle and extend through the first half of the next word.

The format for each 48-bit instruction varies considerably. These will be discussed later. Following is the general form for the 24-bit instructions.



f = function code

b or j = index designator or jump designator

m, y, or k = execution address, where m represents address, y represents operand, and k represents shift count.

The function code is six bits and is different for each instruction.

The index designator represents an index register and is used for address modification. If b=0, no index register is represented and direct addressing is implied. If b=1-6,

one of six index registers is represented and relative addressing is implied. If b=7, indirect addressing is implied. At this time direct addressing (b=0) will be discussed. Later, relative and indirect addressing will be discussed.

The execution address portion is fifteen bits and represents the base address (m), base operand (y), or base shift count (k). However, this is not the actual base address, base operand, or base shift count used. The actual address, operand, or shift count used is M, Y, or K where,

1. $M = m + (B^b)$, the address plus the contents of an index register.
2. $Y = y + (B^b)$, the operand plus the contents of an index register.
3. $K = k + (B^b)$, the shift count plus the contents of an index register.

The index register mentioned is the one specified by b. If b=0, no address modification is performed and,

1. $M = m$
2. $Y = y$
3. $K = k$

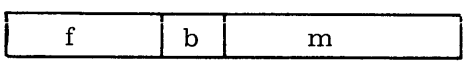
If m is specified in the format of an instruction, it represents a 15 bit address and will require a memory reference in order to read or write a 48 bit word.

If y is specified in the format of an instruction, it represents a 15 bit operand. This instruction transmits the operand y and does not require a memory reference.

If k is specified in the format of an instruction, it represents a 15 bit shift count. This instruction shifts a register by k and does not require a memory reference.

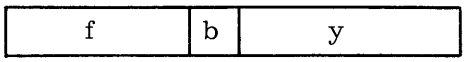
To make this concept clear we will illustrate with an example.

1. Suppose the Load A instruction had the format;



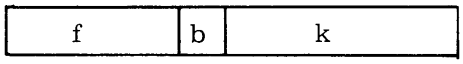
What would 12 0 10000 mean? 12 means Load A with the contents of memory address 10000. One memory reference is made.

2. Suppose the Enter A instruction had the format;



What would 10 0 00126 mean? 10 means Enter A. Index designator b-0 means no modification (direct addressing). 126 means enter A with the octal number 126. No memory reference is made.

3. Suppose the A Right Shift instruction had the format;



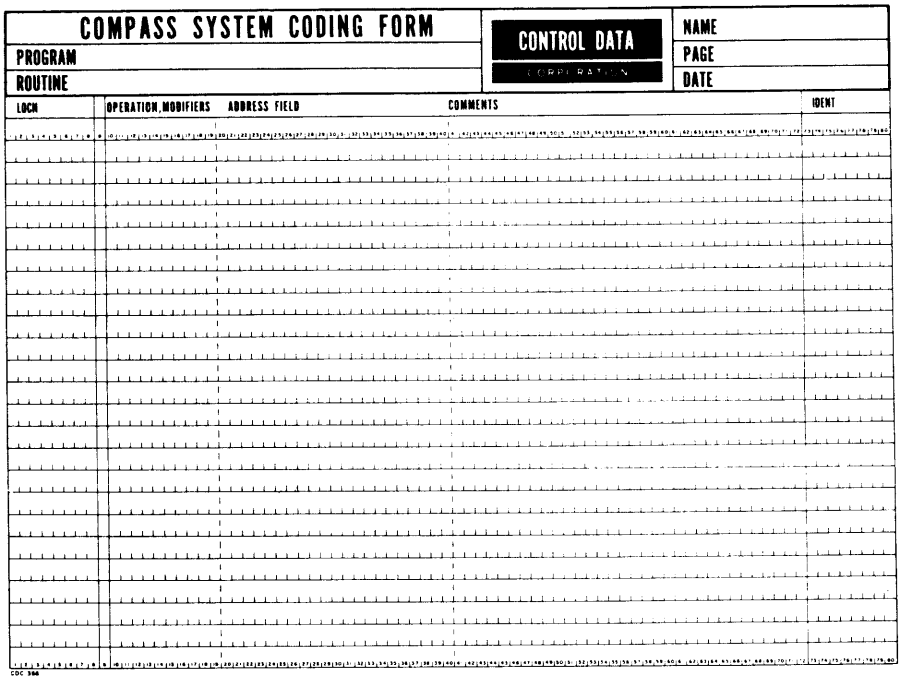
What would 01 0 00030 mean? 10 means A Right Shift. Index designator b-0 means no modification (direct addressing). 30 means shift the contents of A by 30_8 places. No memory reference is made.

These three examples show the difference of using the form m, y, or k in different types of instructions. For the time being we will stay with direct addressing (b=0) treating relative addressing (b=1-6) and indirect addressing (b=7) later.

COMPASS INSTRUCTION FORMAT

At this time we want to relate hardware instruction format to Compass format because all problems will be coded in Compass. On the following page is an example form of the 34/36/3800 Compass coding sheet*.

* Columns 73-80 are purposely left out for all examples in this manual. However, the field is explained.



Note the "fields" on the form.

1. Location field (columns 1-8)
2. Operation field (starting with column 10)
3. Address field (starting with column 20)
4. Comments field (starting with column 41)
5. Ident field (starting with column 73)

Within these fields we enter symbols rather than absolute numbers because symbols are much easier to remember.

Instead of remembering an absolute address like 34175, it is easier to remember a symbolic address like ELEMENT, especially when an operand must be stored and retrieved from this address several times. If we reference address ELEMENT, the

computer assembler will allow an absolute address for this symbol. From this point on we don't care where the address is as long as we can reference it by the symbol.

Instead of remembering 12 as a Load A instruction, it is much easier to remember LDA and let the assembler remember that it represents a 12.

With these notes in mind we now want to discuss the coding form and define a few rules that we need to know before writing a program.

A symbol on the coding form must be eight characters or less. The first character must be a letter rather than a number. Succeeding characters may be letters or numbers, or a mixture of both (alpha-numeric). The following are valid symbols:

1. VALU1
2. A1
3. TESTCASE
4. X1ARRAY

The following are invalid symbols:

1. 4TH
2. TESTAGAIN

Can you tell why?

A symbol on the coding form may not have special characters within it. Characters illegal are: $, *, 1, , , (,), +, and -. A period is allowed.

FIELD RULES

The LOCATION field consists of columns 1-8 which may be all blank or contain any of the following:

1. A legal symbol as defined above (Imbedded blanks are ignored). This symbol usually represents an address.

2. A plus or minus sign placed anywhere in the field with the remaining columns blank. A plus sign forces that particular instruction into the upper half of a memory word. A minus sign forces that particular instruction into the lower half of a memory word.

The OPERATION field starts with column 10 and ends with the first blank column. The Operation field may contain:

1. The mnemonic for the machine language instruction (e.g. LDA)
2. The pseudo instruction (explained later)
3. A macro instruction (explained later)
4. The actual numeric function code (e.g. 12)

There are also sub-fields in the Operation field. However, these will be discussed later.

The ADDRESS field begins in any column following the operation's field blank terminator. It then ends when the first blank character is encountered. For convenience we usually start in column 20 (see dotted line on coding form). We usually start here so that the source listing is easier to read in case the program has to be de-bugged.

The COMMENTS field begins in any column following the address field blank terminator and ends at column 72. For convenience we usually start at column 41 (see dotted line on coding form). We usually start here in order to make the comments easily readable. This field is not interpreted by Compass but only printed on the listing. It can become very valuable, however, to the programmer/analyst in remembering what each section of the program is doing. A programmer should use many brief comments as helpful aids when he initially codes a program. It makes it much more meaningful.

The IDENT field starts with column 72 and ends with column 80. Here a programmer will usually number his cards in ascending order 1 through N so that if he happens to drop the source deck he won't have any trouble rearranging them. This would be quite a task if the cards wouldn't be numbered.

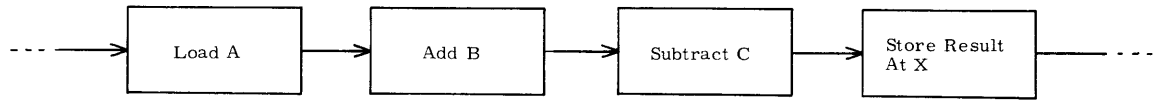
COMPUTER - ORIENTED PROBLEM SOLVING

This portion of Section III deals with problems involving some, but not all of the instructions within the repertoire. The idea is to introduce a few instructions through problem-solving, understand how the instructions are coded, and understand how the instruction set solves the problem. In Vol. II a detailed analysis of the complete instruction repertoire is given along with more problems using them.

Problem 1:

An equation for X states that, X = A + B - C. Solve for X if A, B and C are given.

Flowchart:



The instructions needed to solve this problem are:

1. LOAD A - LDA
2. ADD - ADD
3. SUBTRACT - SUB
4. STORE A - STA

1. Instruction: LOAD A Mnemonic: LDA

Elementary Form:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LDA	m	

This instruction replaces the A register contents with an operand from memory address m. A storage reference is made to obtain the 48-bit quantity. A is cleared, the 48-bit quantity is then loaded into the A register. The memory location remains unchanged.

2. Instruction: ADD Mnemonic: ADD

Elementary Form:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	ADD	m	

This instruction adds a 48-bit operand to the previous contents of the A register in 1's complement, fixed point format. A storage reference is made to obtain the 48-bit quantity at memory address m. The memory location remains unchanged.

When the sum of two quantities exceeds the capacity of A, an arithmetic overflow fault occurs which may cause the computer to be interrupted.

3. Instruction: SUBTRACT Mnemonic: SUB

Elementary Form:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SUB	m	

This instruction subtracts a 48-bit operand from the previous contents of the A register in 1's complement, fixed point format. A storage reference is made to obtain the 48-bit quantity at memory address m. The memory location remains unchanged.

When the difference of two quantities exceeds the capacity of A, an arithmetic overflow fault occurs which may cause the computer to be interrupted.

4. Instruction: STORE A Mnemonic: STA

Elementary Form:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	STA	m	

This instruction stores the contents of the A register at storage location m. The A register contents are not modified by this instruction.

Problem 1 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LDA	A	
	ADD	B	A+B
	SUB	C	A+B-C
	STA	X	ANSWER

The symbols in the address field refer to symbolic addresses which would have to be defined somewhere in the location field.

Student Problem 1A:

An equation for Y states that, Y = A - B - (B + C). Solve for Y if A, B, and C are given.

Flowchart*:

Problem 1A could be solved by coding in the following manner*:

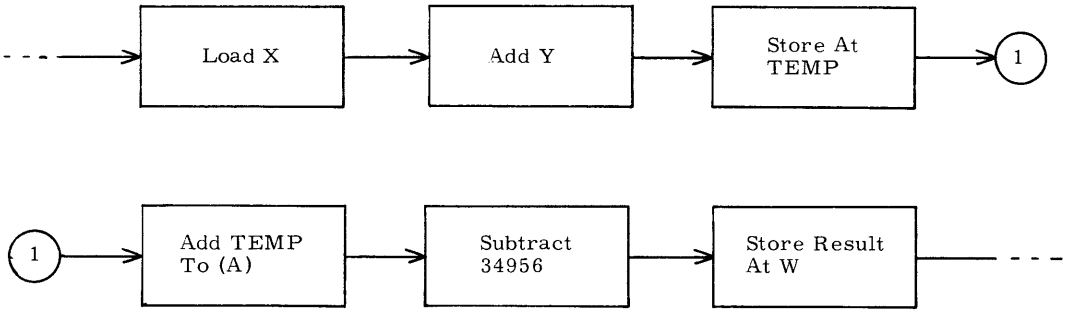
LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS

* The reader is expected to flowchart and provide the coding for all student problems.

Problem 2:

An equation for W states that, W = 2 (X + Y) - 34956. Solve for W if X and Y are given.

Flowchart:



Two new instructions introduced at this time are storage allocation instructions:

1. Block Reservation - BSS
2. Decimal Data Initialization - DEC

1. Instruction: Block Reservation Mnemonic: BSS

Elementary Form:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
ℓ	BSS	n	

This instruction is not a hardware instruction but rather a pseudo instruction. As such it is interpreted by the assembler; not the hardware. When the assembler interprets the instruction, it reserves a block of storage (n words) at this point of the program and the starting address of the block is labeled ℓ (location symbol). When this block is reserved in core, it is not zeroed out. Usually the programmer will store data in the reserved area using it as temporary storage and then read out the data later in the program.

2. Instruction: Decimal Data Initialization Mnemonic: DEC

Elementary Form:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
L	DEC	C	

This instruction is not a hardware instruction but rather a pseudo instruction. As such it is interpreted by the assembler; not the hardware. When the assembler interprets the instruction, it prestores a fixed point decimal constant (no point) at the address specified in the location field.

More can be said about this instruction but this will be in Volume II.

Problem 2 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LDA	X	
	ADD	Y	SUM OF X AND Y
	STA	TEMP	
	ADD	TEMP	2 (X+Y)
	SUB	CONSTANT	2 (X+Y) - 34956
	STA	W	ANSWER

Somewhere, but not in this line of coding would exist the following three cards:

TEMP	BSS	1
W	BSS	1
CONSTANT	DEC	34956

Two locations are reserved: One called TEMP and one called W. One location contains the decimal fixed point number, 34956.

Student Problem 2A:

An equation for Y states that, Y = 5413648 - 3(A-B). Solve for Y if A and B are given.

Flowchart:

Problem 2A would be solved by coding in the following manner:

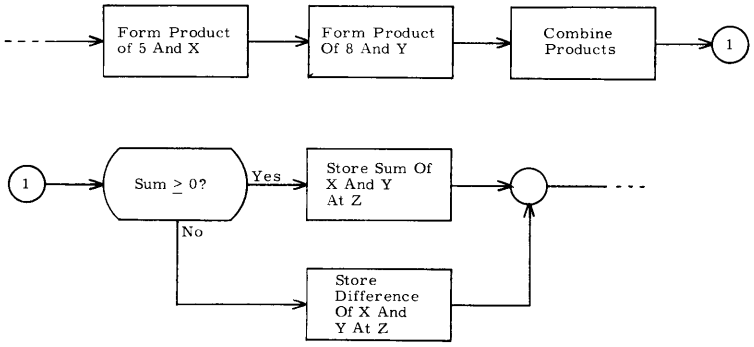
LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

Problem 3:

Given: X and Y as integers each greater than -1600 but less than +1600. For any given values for X and Y:

1. If $5X + 8Y \geq 0$, form the sum of X and Y and store at Z
2. If $5X + 8Y < 0$, form the difference of X and Y and store at Z

Flowchart:



The new instructions needed to solve this problem are:

1. MULTIPLY INTEGER - MUI
2. A JUMP - AJP
3. UNCONDITIONAL JUMP - SLJ/UJP

1. Instruction: MULTIPLY INTERGER Mnemonic: MUI

Elementary Form:

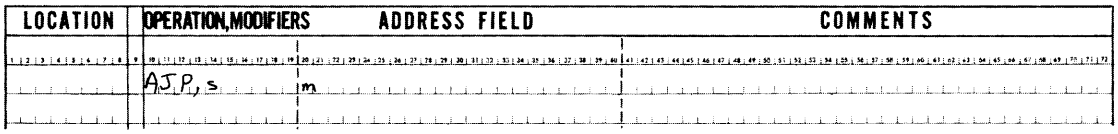
LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	MUI	m	

This instruction forms a 96-bit product from two 48-bit operands. The multiplier must be loaded into the A register prior to the execution of this instruction. The multiplicand is read from storage address m. The resulting product is contained in the QA register as a 96-bit quantity where least significant bits are in A. The contents of address m are not affected by this instruction.

Since the product resides in a 96-bit register (QA), there is no chance of an overflow condition and no interrupt is possible from the execution of this instruction.

2. Instruction: A JUMP Mnemonic: AJP

Elementary Form:



This is the first instruction encountered with a sub-op modifier specified in the operation field. This sub-op is necessary.

The forms this instruction may take are:

1.	AJP, ZR	m
2.	AJP, NZ	m
3.	AJP, PL	m
4.	AJP, MI	m

This instruction examines the contents of the A register for one of four conditions; zero, non-zero, positive, or negative. Hence the reason and necessity for the sub-op modifiers. If the condition checked for is satisfied, a jump occurs to address m. If the condition is not satisfied, the next instruction is executed.

3. Instruction: UNCONDITIONAL JUMP Mnemonic: SLJ or UJP

Elementary Form:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	SLJ	m	

This instruction causes an unconditional jump to address m.

Problem 3 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LDA	CØN1	
	MUI	X	5X
	STA	TEMP	
	LDA	CØN2	
	MUI	Y	8Y
	ADD	TEMP	5X + 8Y
	AJP, PL	SUM	JUMP IF SUM ≥ 0
DIFF	LDA	X	
	SUB	Y	X-Y
	STA	Z	
	SLJ	CØNTINUE	
SUM	LDA	X	
	ADD	Y	X+Y
	STA	Z	
CØNTINUE			

Somewhere, but not in this line of coding, would exist the following pseudo instructions:

CON1	DEC	5
CON2	DEC	8
TEMP	BSS	1
Z	BSS	1

Student Problem 3A:

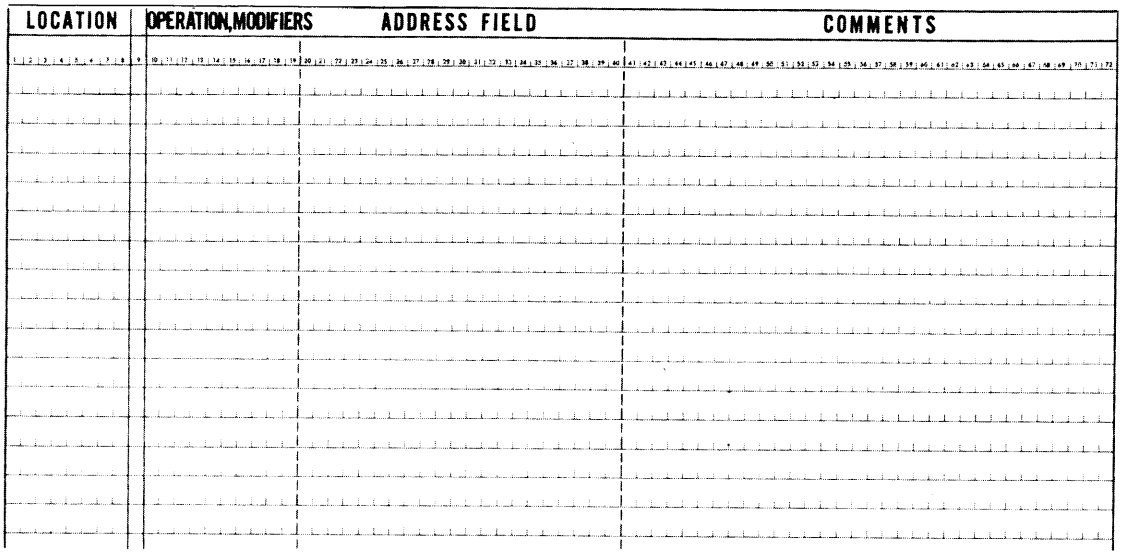
Given: A and B as integers each greater than -1200 but less than +1200. For any given values for A and B:

1. If 10A - 3B = 0, substitute the current values of A and B into this equation Y = 10A + 7B solving for Y.

2. If 10A - 3B $\neq$ 0, jump to address TRYAGN.

Flowchart:

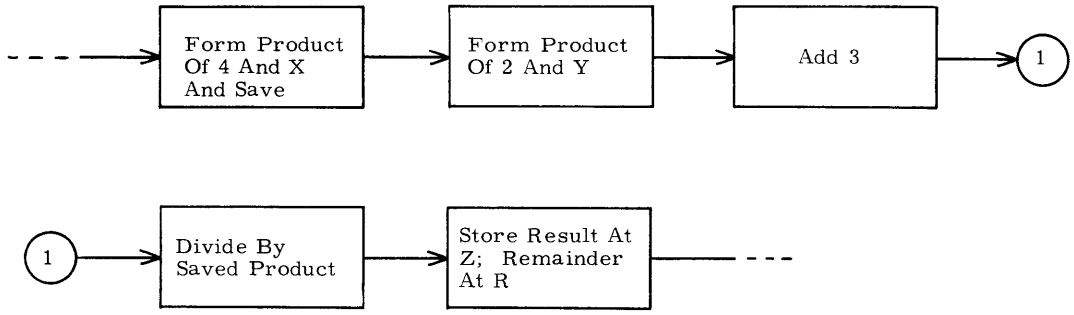
Problem 3A could be solved by coding in the following manner:



Problem 4:

Evaluate $Z = \frac{2Y + 3}{4X}$ if X and Y are given as positive or negative integers between -2000 and +2000. Any remainder is to be stored at R.

Flowchart:



The new instructions needed to solve this problem are:

1. Store Q - STQ
2. Divide Integer - DVI
3. Enter Q - ENQ
4. Increase A - INA

1. Instruction: STORE Q Mnemonic: STQ

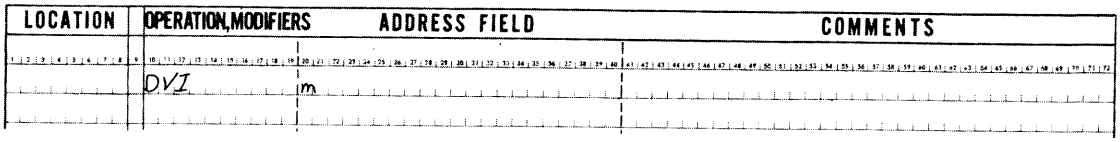
Elementary Form:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	STQ	m	

This instruction stores the contents of the Q register at storage address m. The Q register contents are not modified by this instruction.

2. Instruction: DIVIDE INTEGER Mnemonic: DVI

Elementary Form:



This instruction divides a 96-bit integer dividend by a 48-bit integer divisor. The 96-bit dividend must be formed in the QA register, with the least significant bits in A, prior to the execution of this instruction. The quotient is formed in the A register. The remainder is left in the Q register at the end of the operation. The remainder will have the same algebraic sign as the initial dividend.

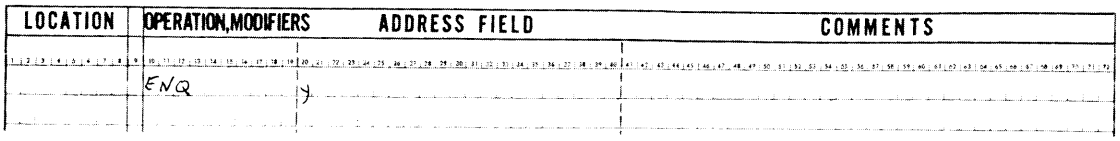
The only fault that can occur as a result of this instruction is a divide fault. This will occur when one of two conditions exist. They are:

1. division by zero
2. the dividend being so large as compared to the divisor that the answer exceeds the capacity of the A register.

A divide fault constitutes an interruptible condition and could force the computer into interrupt.

3. Instruction: ENTER Q Mnemonic: ENQ

Elementary Form:

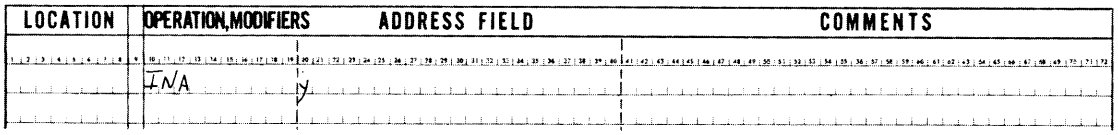


This instruction clears the contents of the Q register and enters it with the 15-bit quantity y, right justified, with the sign (bit 14) of y extended into the upper order bits of Q. No storage reference is made.

Positive or negative numbers are allowed for y up to $\pm 2^{14} - 1$, i.e., $\pm$ 16,383.

4. Instruction: INCREASE A Mnemonic: INA

Elementary Form:



This instruction adds a 15-bit quantity to the A register. Positive or negative numbers are allowed for y up to $\pm 2^{14} - 1$, i.e., $\pm$ 16,383. The addition is performed as if the 15 bits were 48 bits with the 15 bits right justified and the upper order bits simply an extension of the sign bit. The arithmetic is performed in 1's complement notation.

If the operation causes the capacity of the A register to be exceeded, an arithmetic overflow fault will occur which could force the computer into interrupt.

Problem 4 could be solved by coding in the following manner:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	LDA	X	
	MUI	CØN1	4X
	STA	TEMP	
	LDA	Y	
	MUI	CØN2	2Y
	INA	3	
	AJP, PL	PØS	CHECK SIGN
	ENQ	-0	EXTEND NEGATIVE SIGN THROUGH Q
	UJP	RDY	
PØS	ENQ	0	EXTEND POSITIVE SIGN THROUGH Q
RDY	DVI	TEMP	(2Y+3) / (4X)
	STA	Z	
	STQ	R	

Somewhere, but not in this line of coding would exist the following pseudo instructions:

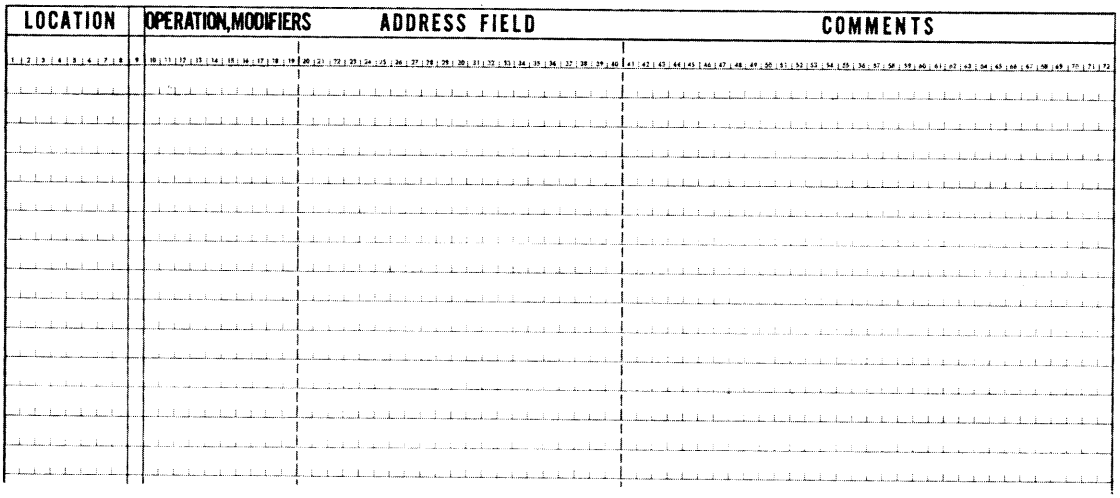
CON1	DEC	4
CON2	DEC	2
TEMP	BSS	1
Z	BSS	1
R	BSS	1

Student Problem 4A:

Evaluate $N = \frac{20I-53}{3J+4K}$ if I, J, and K are given as positive or negative integers between -3000 and +3000. Any remainder after the divide is to be stored at R.

Flowchart:

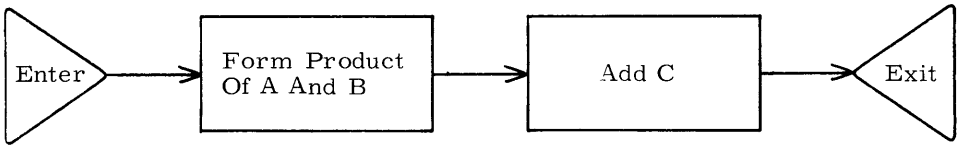
Problem 4A could be solved by coding in the following manner:



Problem 5:

Assuming a Compass subprogram to be executed, code the instructions needed to form A· B + C in the A register if A, B, and C are given as integers between -4000 and +4000.

Flowchart:



The new instructions needed to solve this problem are:

1. SUBPROGRAM IDENTIFICATION - IDENT
2. SUBPROGRAM TERMINATION - END
3. ENTRY POINT - ENTRY

1. Instruction: SUBPROGRAM IDENTIFICATION Mnemonic: IDENT

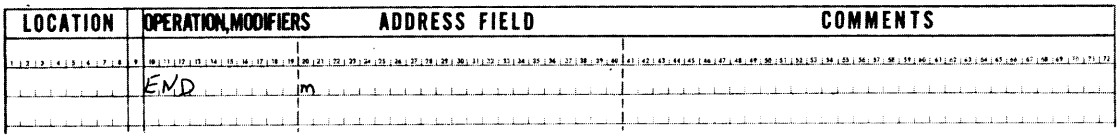
Elementary Form:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	m	

This instruction is a pseudo instruction which must be the first instruction of each subprogram. The m subfield must contain a symbol of 1 to 8 characters, which names the subprogram.

2. Instruction: SUBPROGRAM TERMINATION Mnemonic: END

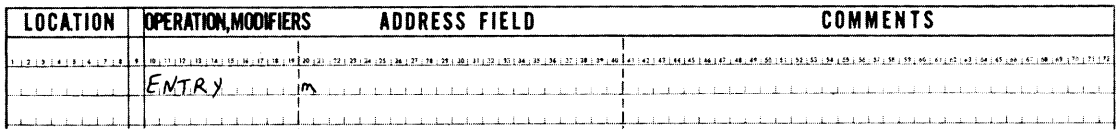
Elementary Form:



This instruction is a pseudo instruction which must be the last instruction of each subprogram. END signals termination of the subprogram. The symbol in the address field is optional. If present, the symbol represents the transfer address to which control is sent when beginning the execution of the program. The symbol must also be declared as an ENTRY point in the subprogram in which it resides.

3. Instruction: ENTRY POINT Mnemonic: ENTRY

Elementary Form:



This instruction is a pseudo instruction that declares the symbol within the subprogram as an entry point that may be referenced by other subprograms. The entry point symbol must be in the location field within the same subprogram that declares it.

The ENTRY instruction does not use up a memory location and can be placed anywhere in the subprogram deck.

Problem 5 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	EVALUATE	
	ENTRY	EVALUATE	
EVALUATE	BSS	1	
	LDA	A	
	MUI	B	AB
	ADD	C	AB + C
	SLJ	EVALUATE	
	END		

The subprogram consists of the instructions IDENT through END which could be entrant from the Scope monitor or from some other subprogram within the job. Entrance would be by means of a "return" jump to EVALUATE. After the algebraic expression is solved, a jump is made to the entry point EVALUATE. The instruction executed at EVALUATE "returns" program control to the calling subprogram. You might keep this idea in mind because a further discussion will follow that will involve the return jump instructions and the use of more than one subprogram.

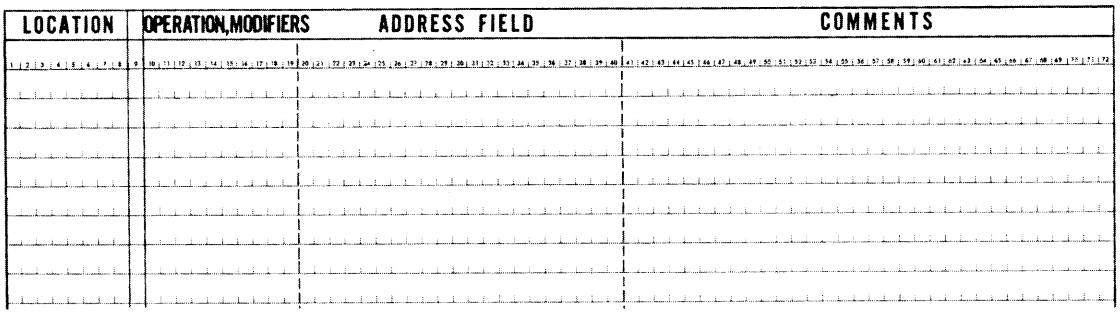
From this point on all problems will be solved by also including the three pseudo instructions just learned: IDENT, ENTRY, and END.

Student Problem 5A:

Use the pseudo instructions just learned to form a subprogram that, when entered, will form $X^2 + X - 1$ in the A register if X is given as an integer between -6000 and +6000.

Flowchart:

Problem 5A could be solved by coding in the following manner:



Problem 6:

Given: 100 scores from a test. Each score ranges from 50 through 100 and takes up one memory location starting at the symbolic address SCORES. Find the average of these scores and store it at AVE. Any remainder store at REM.

There are two ways of going about solving this problem. One way is to add the 100 scores with sequential ADD instructions. Another way is to repeat one ADD instruction until the 100 scores are added. The first method is cumbersome because so many cards have to be punched. The second method takes far fewer cards and uses the concept of INDEXING. It is at this point that we wish to introduce this concept.

At the beginning of this section we presented the machine and Compass format for the 24-bit instructions. We mentioned that the 3 bit index designator b could be used in any of the following manners:

1.	b = 0	no indexing specified
2.	b = 1 - 6	indexing specified
3.	b = 7	indirect addressing

Number 2 is now considered. There exists within the central processor 6 index registers (hence b = 1 - 6) the contents of which could be used as "pointers" that point either forward or backward from the base address of an instruction. This is called relative addressing. The actual address referenced by an instruction always takes into consideration the possibly of indexing. A few examples will show how indexing is performed.

Suppose $(B^1) = 3$, $(B^2) = 5$ and the following instructions were executed.

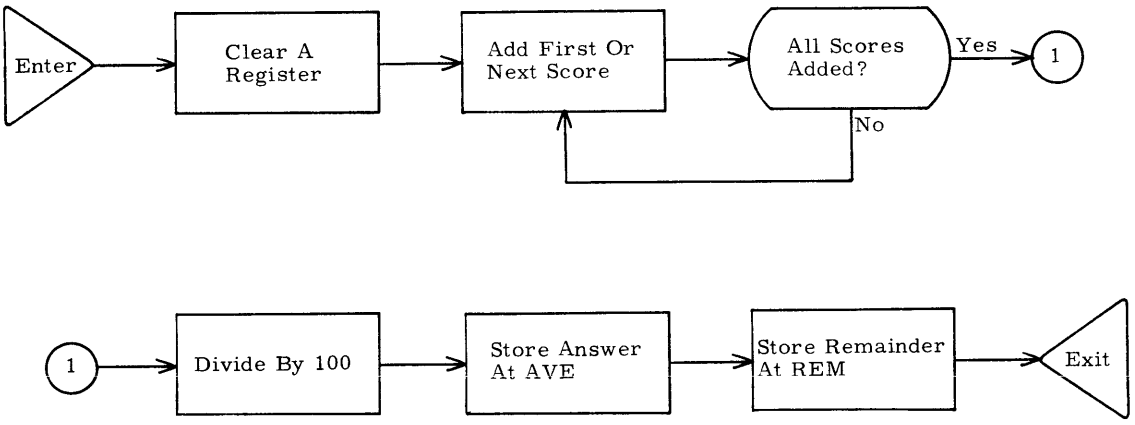
```
1.    LDA     TAX,1
2.    ADD     MAX,2
```

The first instruction loads a 48 bit operand from an address. What address is it? It would be address TAX + 3 (three addresses ahead of address TAX), since index register 1 is specified and it contains +3.

The second instruction adds a 48 bit operand from an address to the contents of A. What address is it? It would be address MAX + 5 (five addresses ahead of address MAX), since index register 2 is specified and it contains +5.

If these instructions are executed repeatedly and each time the contents of the index register are incremented by 1, each address referenced will be one ahead of the address previously referenced. This is the technique we will use to solve the problem.

Flowchart:

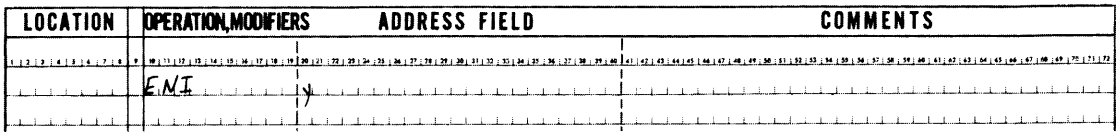


The new instructions needed to solve this problem are:

1. ENTER INDEX - ENI
2. INDEX SKIP - ISK

1. Instruction: ENTER INDEX Mnemonic: ENI

Elementary Form:

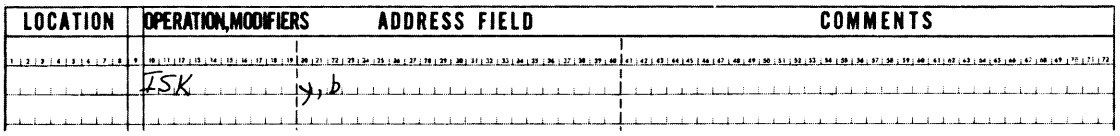


This instruction replaces the contents of the designated index register with the value y . No storage reference is made with this instruction.

The range for y varies from -16,383 through +16,383.

2. Instruction: INDEX SKIP Mnemonic: ISK

Elementary Form:



This instruction compares the quantity in the designated index register with the base operand y. If the two are not equal, a half-exit is taken and the contents of the designated index register are increased by 1 (2's complement). If the two are equal, a full exit is taken and the contents of the designated index register are cleared.

Since this instruction has the capability of half-exiting or full-exiting, it is necessary that this instruction be in the upper half of a memory word. The Compass assembler will automatically force this instruction upper.

Problem 6 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	TEST	
	ENTRY	TEST	
TEST	BSS	1	
	ENI	0,1	
	ENA	0	
REPEAT	ADD	SCØRES,1	
	ISK	99,1	
	SLJ	REPEAT	
	ENQ	0	PREPARE FØR DIVIDE
	DVI	DEC100	
	STA	AVE	
	STQ	REM	
	SLJ	TEST	RETURN
DEC100	DEC	100	
AVE	BSS	1	
REM	BSS	1	
	END		

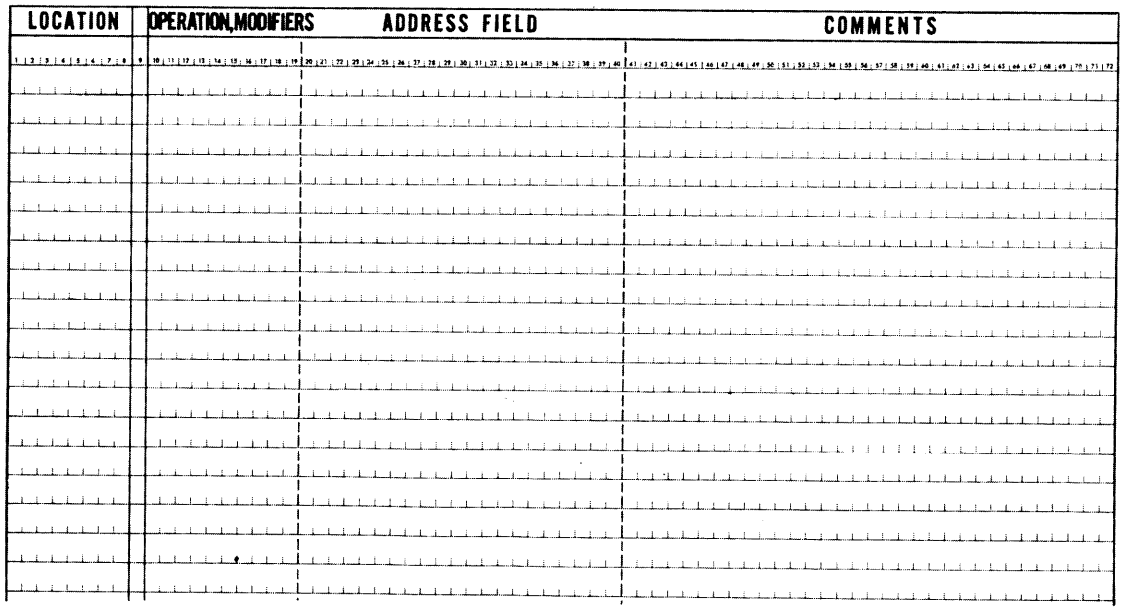
Somewhere within this subprogram would also be included the symbol SCORES in the location field with the 100 scores entered.

Student Problem 6A:

One thousand Army recruits were given a test, their integer scores were entered onto cards, and the cards were read into memory starting at address TALLY. Find the average of these scores and store at MEAN. Any remainder store at REM.

Flowchart:

Problem 6A could be solved by coding in the following manner:



Problem 7:

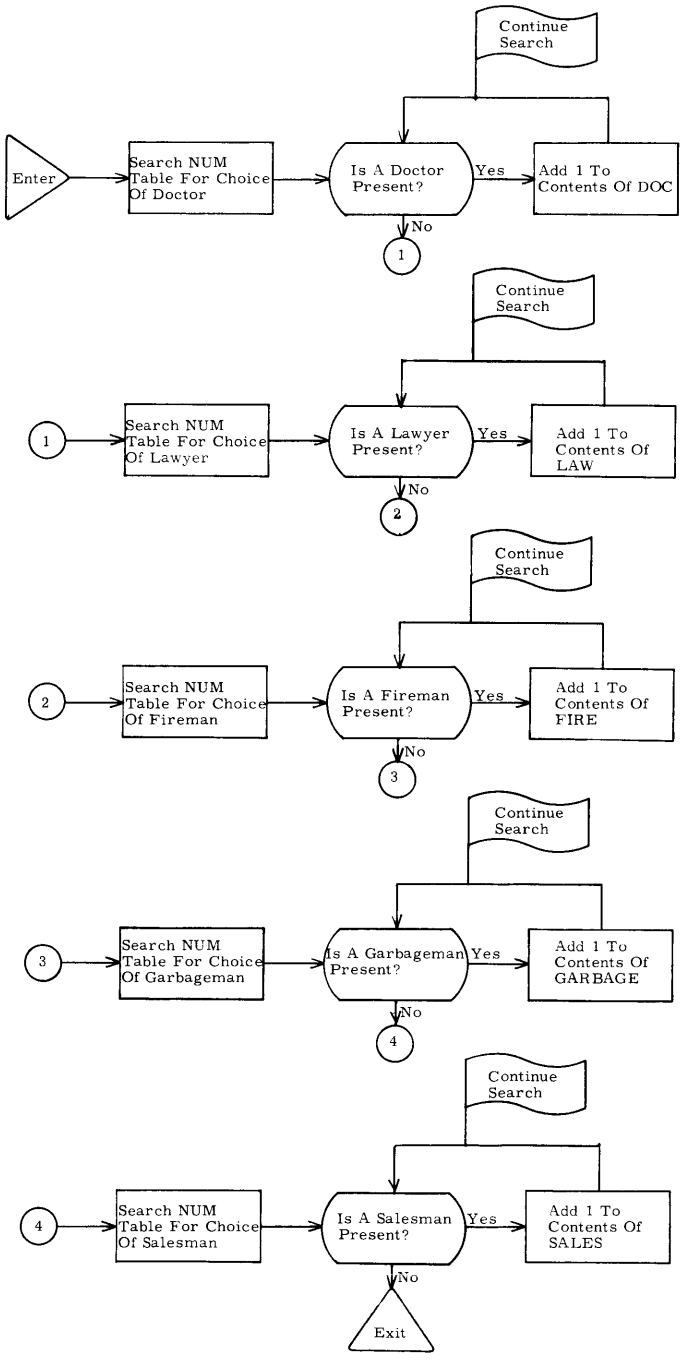
During the school year a group of 10,000 grade school boys were asked which of 5 professions they thought they would be interested in. The 5 professions were:

1. Doctor
2. Lawyer
3. Fireman
4. Garbageman
5. Salesman

Each boy indicated his choice by choosing the number associated with the profession; 1 for doctor, 2 for lawyer, 3 for fireman, 4 for garbageman, and 5 for salesman. The 10,000 numbers were entered into the computer starting at address NUM, each number taking up one memory location.

From the list of 10,000 numbers determine how many boys thought they would like to be doctors (indicate number in address DOC), how many would like to be lawyers (indicate number in address LAW), and how many would like to be fireman (indicate number in address FIRE), how many would like to be garbagemen (indicate number in address GARBAGE), and how many would like to be salesmen (indicate number in address SALES).

Flowchart:

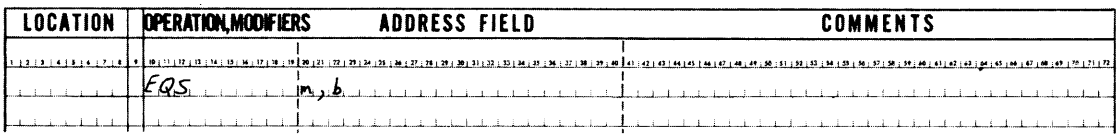

Continue Search
Enter
Search NUM Table For Choice Of Doctor
Is A Doctor Present?
Yes
Add 1 To Contents Of DOC
No
1
Continue Search
1
Search NUM Table For Choice Of Lawyer
Is A Lawyer Present?
Yes
Add 1 To Contents Of LAW
No
2
Continue Search
2
Search NUM Table For Choice Of Fireman
Is A Fireman Present?
Yes
Add 1 To Contents Of FIRE
No
3
Continue Search
3
Search NUM Table For Choice Of Garbageman
Is A Garbageman Present?
Yes
Add 1 To Contents Of GARBAGE
No
4
Continue Search
4
Search NUM Table For Choice Of Salesman
Is A Salesman Present?
Yes
Add 1 To Contents Of SALES
No
Exit

The new instructions needed to solve this problem are:

1. EQUALITY SEARCH - EQS
2. REPLACE ADD ONE - RAO

1. Instruction: EQUALITY SEARCH Mnemonic: EQS

Elementary Form:



This instruction searches a list of consecutive operands to find one that is equal to the contents of the A register. If an operand within the list satisfies the search, a full exit is taken skipping the lower instruction. If no operand within the list satisfies the search, a half exit is taken.

Before the instruction is executed, the specified index register must contain the number of words to be examined. For each operand examined the contents of the index designation are reduced by 1. The designator m is the first word address of the list. If no index designator is specified, one word is examined, at address m.

If an operand satisfies the search, the address of that operand is given as $M + (B^b)$. If the instruction is re-entered without destroying the contents of either the specified index register or A, the search continues with the next location. This technique is used in the coding for the solving of the problem.

2. Instruction: REPLACE ADD ONE Mnemonic: RAO

Elementary Form:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RAØ	m	

This instruction replaces the quantity at address m with its original value plus one in integer format. The resultant sum is also left in the A register.

Problem 7 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	INTEREST	
	ENTRY	INTEREST	
DØC	DEC	0	PRESTØRE ZERØS
LAW	DEC	0	AT THESE
FIRE	DEC	0	LØCATIØNS
GARBAGE	DEC	0	
SALES	DEC	0	
INTEREST	BSS	1	
	ENI	10000,1	
	ENA	1	
A	~~ENQ~~ EQS	NUM,1	SEARCH FØR DØCTØR
	SLJ	B1	NØ MØRE DØCTØRS
	RAØ	DØC	FØUND A DØCTØR
	SLJ	A	CØNTINUE SEARCH
B1	ENI	10000,1	
	ENA	2	
B2	EQS	NUM,1	SEARCH FØR LAWYERS
	SLJ	C1	NØ MØRE LAWYERS
	RAØ	LAW	FØUND A LAWYER
	SLJ	B2	CØNTINUE SEARCH
C1	ENI	10000,1	
	ENA	3	
C2	EQS	NUM,1	SEARCH FØR FIREMEN
	SLJ	D1	NØ MØRE FIREMEN
	RAØ	FIRE	FØUND A FIREMAN
	SLJ	C2	CØNTINUE SEARCH
D1	ENI	10000,1	
	ENA	4	
D2	EQS	NUM,1	SEARCH FØR GARBAGEMEN
	SLJ	E1	NØ MØRE GARBAGEMEN
	RAØ	GARBAGE	FØUND A GARBAGEMAN
	SLJ	D2	CØNTINUE SEARCH
E1	ENI	10000,1	
	ENA	5	
E2	EQS	NUM,1	SEARCH FØR SALESMEN
	SLJ	INTEREST	NØ MØRE SALESMEN
	RAØ	SALES	FØUND A SALESMAN
	SLJ	E2	CØNTINUE SEARCH
	END		

Somewhere within this subprogram would also be included the symbol NUM in the location field with a declaration of the 10000 memory locations

Student Problem 7A:

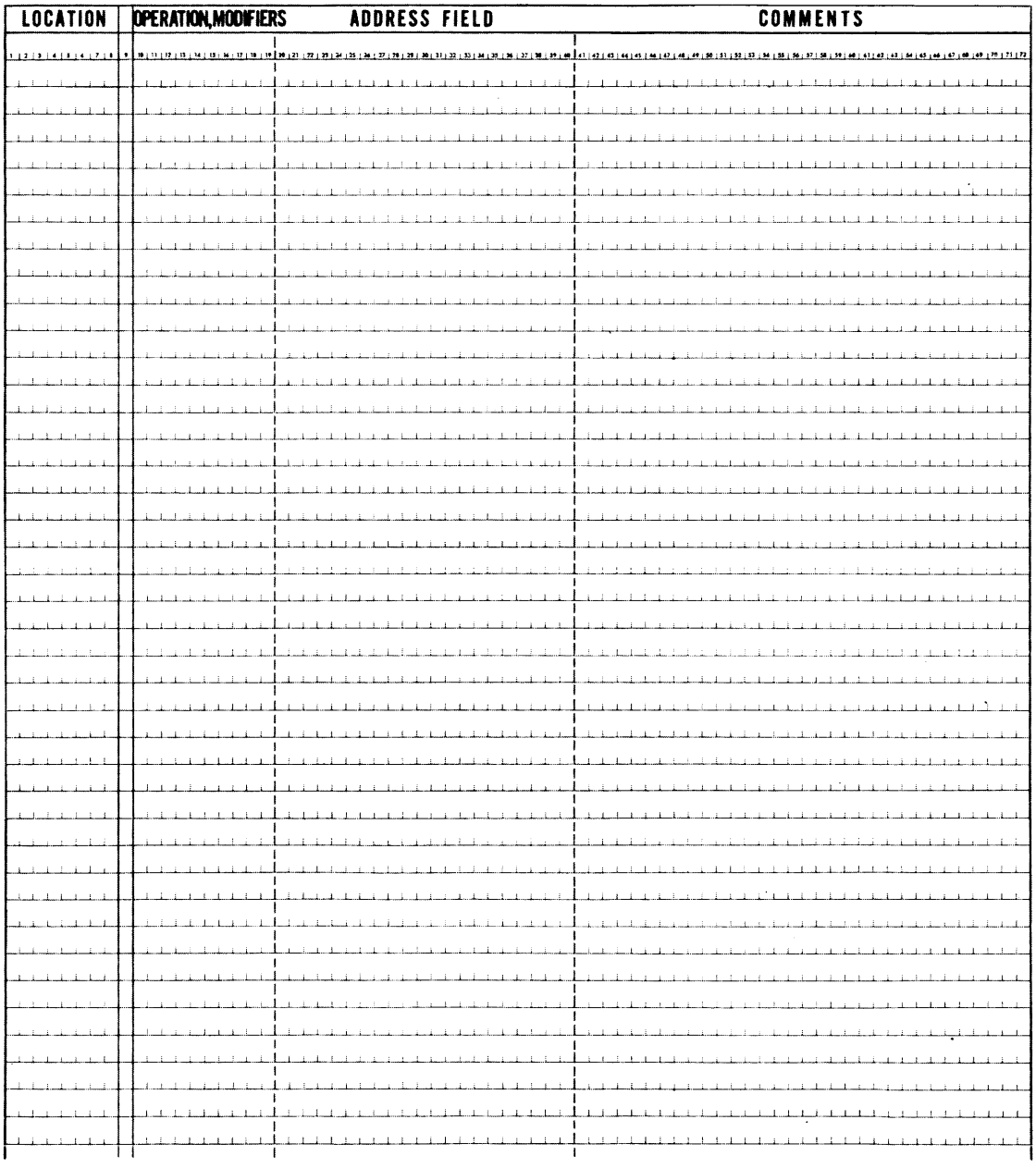
A group of 943 students took a final wxam and were given points for each grade in the following manner:

1. A = 4 points
2. B = 3 points
3. C = 2 points
4. D = 1 point
5. F = 0 points

All grades were entered into the computer, each grade represented by its associated value, and each grade taking up one memory location. Write a program that will indicate the number of students failing the exam (F). Store this number at address FLUNK.

Flowchart:

Problem 7A could be solved by coding in the following manner:



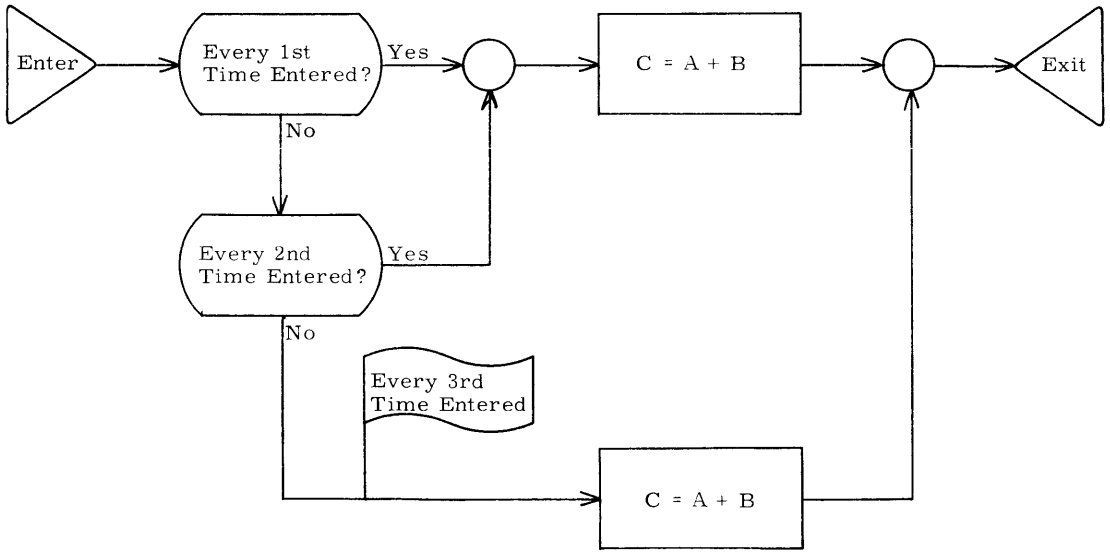
Problem 8:

Write a Compass subprogram such that every 1st and 2nd time it is entered it calculates C = A + B, and every 3rd time it is entered it calculates C = A - B. The series would look like the following:

1. C = A + B
2. C = A + B
3. C = A - B
4. C = A + B
5. C = A + B
6. C = A - B

.
.
.

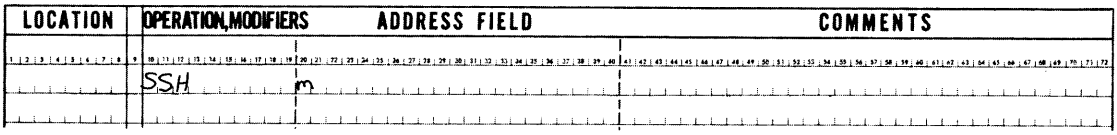
Flowchart:



The new instructions needed to solve this problem are:

1. STORAGE SHIFT - SSH
2. OCTAL DATA INITIALIZATION - OCT

1. Instruction: STORAGE SHIFT Mnemonic: SSH

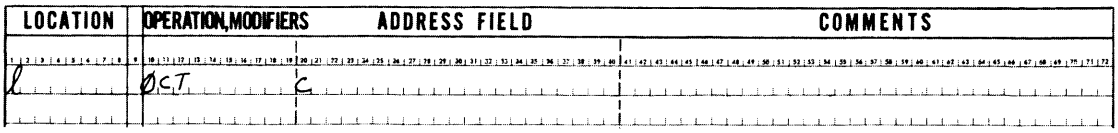


This instruction senses the sign bit of the quantity at storage address m. If the sign of the quantity is negative, a full exit is taken skipping the next instruction. If the sign of the quantity is positive, a half exit is taken and no instruction is skipped.

In either case the quantity in storage is then left shifted end-around one binary bit position. The A and Q registers are not affected by this instruction.

Since this instruction has the capability of half or full exiting upon the sensing, it is automatically placed in the upper half of the memory word by the Compass assembler.

2. Instruction: OCTAL DATA INITIALIZATION Mnemonic: OCT
Elementary Form:



This instruction is closely related to the DEC instruction. Hence it is not a hardware instruction but rather a pseudo instruction. As such it is interpreted by the assembler. When the assembler interprets it, it prestores a fixed point octal constant at the address specified in the location field. The constant C may be up to 16_{10} digits since this is the maximum at any address.

Problem 8 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	ENTRANT	
	ENTR	ENTRANT	
C	BSS	1	
CHECK	OCT	1111111111111111	
ENTRANT	BSS	1	
	SSH	CHECK	1ST, 2ND, OR 3RD?
	SLJ	APLB	1ST OR 2ND
	LDA	A	3RD
	SUB	B	
	STA	C	C = A - B
	SLJ	ENTRANT	
APLB	LDA	A	
	ADD	B	
	STA	C	C = A + B
	SLJ	ENTRANT	
	END		

Somewhere within the subprogram would also be included the symbols A and B in the location field with a declaration of the prestored data or area reserved.

Student Problem 8A:

Write a subprogram that will calculate Y is sometimes Y = 2A end sometimes Y = -3B according to the following pattern:

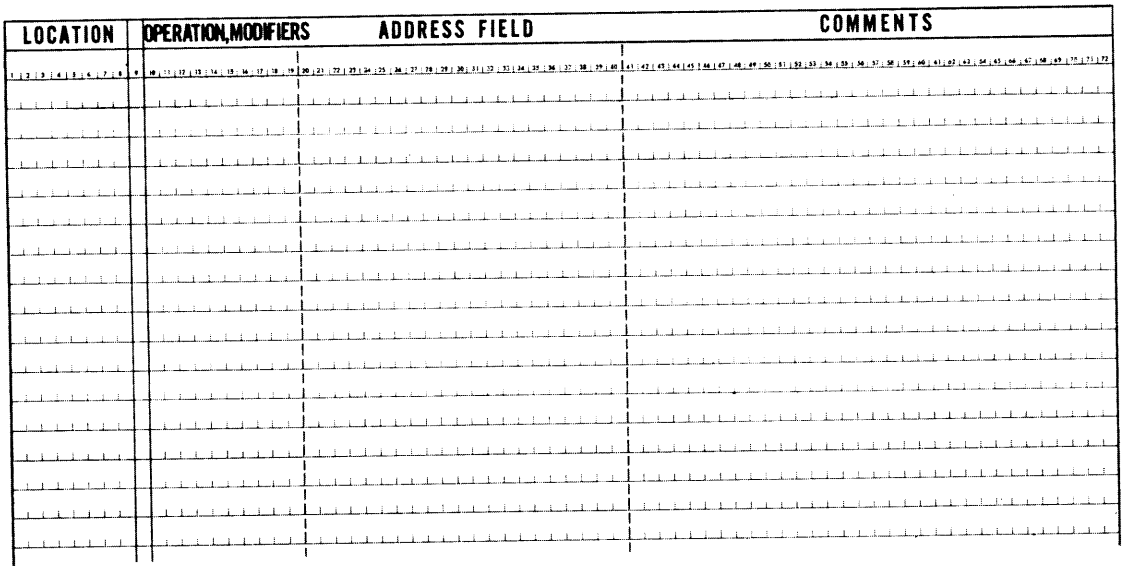
1. Y = 2A
2. Y = 2A
3. Y = -3B
4. Y = 2A
5. Y = -3B
6. Y = -3B
7. Y = 2A
8. Y = 2A
9. Y = -3B
10. Y = 2A
11. Y = -3B
12. Y = -3B

.
.
.

A and B are given as integers between -30, 000 and +30, 000.

Flowchart:

Problem 8A could be solved by coding in the following manner:



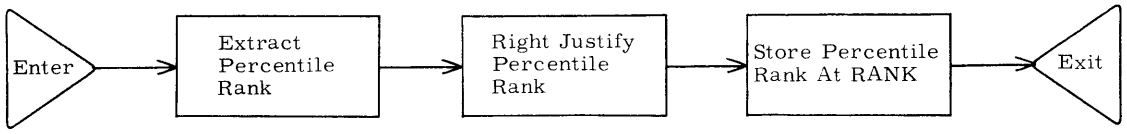
Problem 9:

A group of 5000 senior high school students took a college entrance examination and were told to remember an assigned number if they wanted to find out pertinent information on how well they did. The information was stored in memory in the form of a block of 5000 words starting at address INFORM, each word representing information on a particular student. The block looked like the following:

Address		Word
(INFORM)	=	Pertinent information on student #0
-		-
-		-
-		-
-		-
-		-
-		-
(INFORM+4999)	=	Pertinent information on student #4999

Assuming a student's percentile ranking to be positioned at bit positions 24 through 32 of each word, write a subprogram that will store a student's percentile ranking right justified at address RANK if his number initially is in index register 1.

Flowchart:

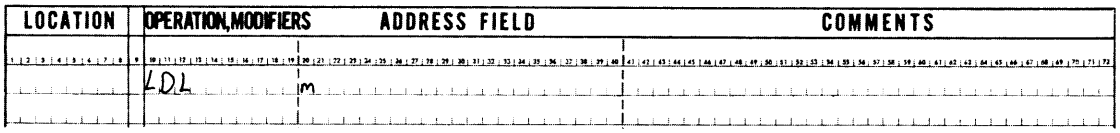


The new instructions needed to solve this problem are:

1. LOAD LOGICAL - LDL
2. A RIGHT SHIFT - ARS
3. Q LEFT SHIFT - QLS

1. Instruction: LOAD LOGICAL Mnemonic: LDL

Elementary Form:



This instruction loads the A register with the bit by bit logical product of the Q register contents and the quantity in the referenced storage address.

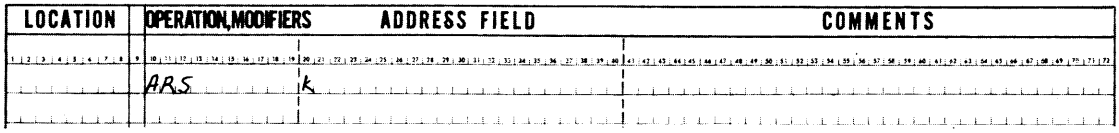
By "bit by bit logical product" we mean the corresponding bits under multiplication. Illustrated are the four cases.

1st Operand	0	0	1	1
2nd Operand	0	1	0	1
Logical Product	0	0	0	1

The Q register frequently contains a mask which must be loaded prior to the execution of this instruction. The mask consists of a set of "1" bits: "1" multiplied by any bit will return the bit. "0" bits are used to eliminate unwanted bits since "0" multiplied by any bit will always return zeros.

2. Instruction: A RIGHT SHIFT Mnemonic: ARS

Elementary Form:



This instruction shifts the contents of the A register to the right by k number of bit positions. The sign bit is extended as the bits shift right. The lowest order bits are discarded as they are shifted out of the register (end-off).

3. Instruction: Q LEFT SHIFT Mnemonic: QLS

Elementary Form:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	QLS	k	

This instruction shifts the contents of the Q register to the left by k number of bit positions. The highest order bits are shifted circularly and are picked up at the lower end of the register (end around). No bits are lost during a normal left shift.

Problem 9 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	ENTREXAM	
	ENTRY	ENTREXAM	
RANK	BSS	1	
ENTREXAM	BSS	1	
	ENQ	777B	ØCTAL 777
	QLS	24	
	LDL	INFORM,1	
	ARS	24	
	STA	RANK	
	SLJ	ENTREXAM	
	END		

Somewhere within the subprogram would also be included the symbol INFORM in the location field with a declaration of the prestored data or area reserved.

Student Problem 9A:

Assume the same situation in the previous problem. If a student's IQ rating is positioned at bit positions 0 through 8 of each word, write a subprogram that will store his IQ rating right justified at address IQ. Assume the student's number initially in index register 1.

Flowchart:

Problem 9A could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

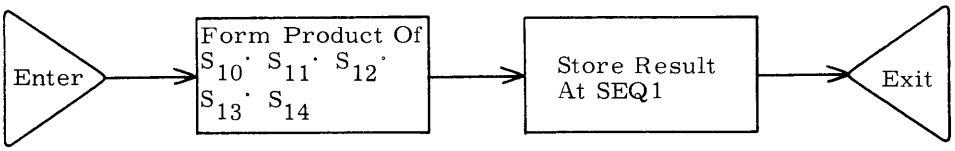
Problem 10:

A sequence of terms looks like the following:

1/2, 2/3, 3/4, 4/5 . . .

Assume this sequence of terms is in core starting at address SEQ and that each fraction is in fractional format, one memory location per fraction. Write a subprogram that will multiply the 10th through 14th terms with the answer stored at address SEQ1.

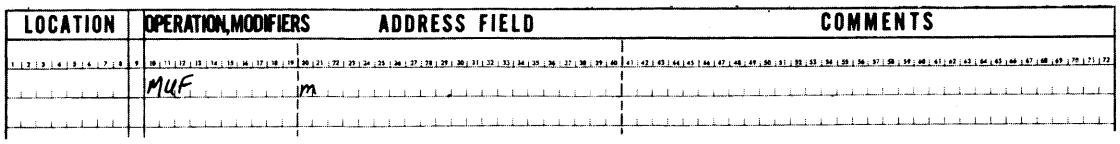
Flowchart:



The new instruction needed to solve this problem is:

1. MULTIPLY FRACTIONAL - MUF

1. Instruction: MULTIPLY FRACTIONAL Mnemonic: MUF
Elementary Form:



This instruction forms a 96-bit product from two 48-bit operands. All quantities in this operation are treated as fractions with the binary point immediately to the right of the sign bit. The sign bit is again the leftmost bit of the operand as we have in the integer format.

The multiplier must be loaded into the A register prior to the execution of the instruction. The multiplicand is read from the storage location specified by the sum of the execution address and the contents of the designated index register. The product is formed in the AQ register.

Since this instruction treats both initial operands as fractions, the answer in AQ is also a fraction that is less than either the multiplier or multiplicand.

No error can result from this instruction barring a machine malfunction.

Problem 10 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	SEQUENCE	
	ENTRY	SEQUENCE	
SEQ1	BSS	1	
SEQUENCE	BSS	1	
	ENI	0,1	
	LDA	SEQ+9	10TH TERM
REPEAT	MUF	SEQ+10,1	
	ISK	3,1	
	SLJ	REPEAT	
	STA	SEQ1	
	SLJ	SEQUENCE	
	END		

Somewhere within the subprogram would also be included the symbol SEQ in the location field with a declaration of the prestored data.

Student Problem 10A:

A sequence of terms consists of the following:

+ 2/3, - 4/5, + 6/7, - 8/9, + 10/11, . . .

If this sequence starts at address NUM and is in fractional format, write a subprogram to calculate the product of the 20th through 25th term and store the result at address RES.

Flowchart:

Problem 10A could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS

Problem 11:

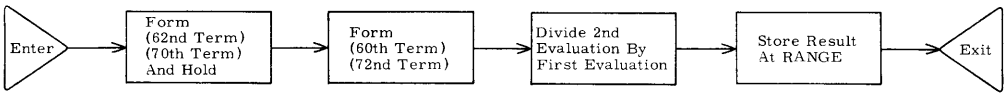
A sequence of terms resides in memory starting at address TERM according to the pattern:

1/2, 2/3, 4/5, 5/6, 7/8, 8/9, . . .

If each fraction is in fractional format and each takes up one memory location, write a subprogram that will form at address RANGE:

RANGE = ((60th Term) (72nd Term)) / ((62nd Term) (70th Term))

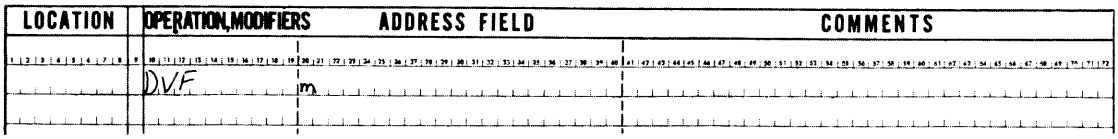
Flowchart:



The new instruction needed to solve this problem is:

1. DIVIDE FRACTIONAL - DVF

1. Instruction: DIVIDE FRACTIONAL Mnemonic: DVF
Elementary Form:



This instruction divides a 96-bit dividend by a 48-bit divisor. All quantities involved in this operation are treated as fractions with the binary point immediately to the right of the sign bit. The 96-bit dividend must be loaded into the AQ register prior to the execution of this instruction. The 48-bit divisor is read from storage address m. At the end of the operation, the quotient is left in the A register and the remainder is left in the Q register. The quotient and remainder bear the same algebraic sign.

A divide fault is easily attained using this instruction. Any attempt to divide by zero will yield this condition. Also, anytime an attempt is made when the dividend is equal to or greater than the divisor, a divide fault will occur. Besides the initial operands being fractional the result must be fractional, i.e., only fractions are allowed.

A divide fault constitutes an interruptible condition.

Problem 11 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	MATRIX	
	ENTRY	MATRIX	
RANGE	BSS	1	
DIVIS	BSS	1	
MATRIX	BSS	1	
	LDA	TERM+61	= 62ND TERM
	MUF	TERM+69	= 70TH TERM
	STA	DIVIS	
	LDA	TERM+59	= 60TH TERM
	MUF	TERM+71	= 72ND TERM
	DVF	DIVIS	
	STA	MATRIX	RETURN
	END		

Somewhere within this subprogram would also be included the symbol TERM in the location field with a declaration of the prestored area.

Student Problem 11A:

Evaluate $Z = \frac{.9X}{Y}$ if X and Y are fractions and X < Y. Assume .9 is stored at address CON1.

Flowchart:

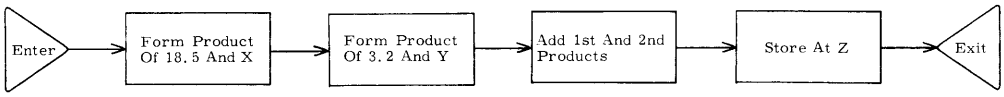
Problem 11A could be solved by coding in the following manner:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
1 2 3 4 5 6 7 8	9 10 11 12 13 14 15 16 17 18 19	20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40	41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72

Problem 12:

Evaluate Z = 18.5 X + 3.2 Y if X and Y are given.

Flowchart:



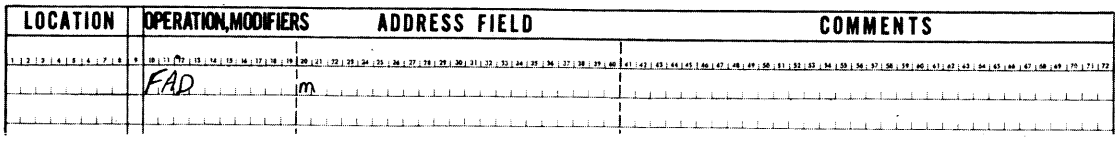
This is the first introduction to floating point instructions. Since mixed numbers are involved, it is necessary to use floating point arithmetic because neither pure integer nor pure fractional arithmetic would suffice. Floating point operations allow integer fractional or mixed numbers to be used. A detailed discussion of floating point format follows in the next section.

The new instructions needed to solve this problem are:

1. FLOATING ADD - FAD
2. FLOATING MULTIPLY - FMU
3. Floating Point Data Initialization - DEC

1. Instruction: FLOATING ADD Mnemonic: FAD

Elementary Form:

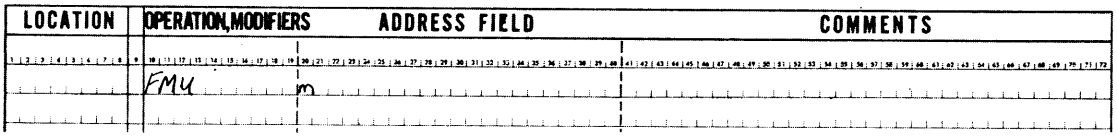


This instruction forms the sum of two 48-bit quantities which are both packed in floating point format. An operand is read from the storage location specified by m and is added to the previous contents of A. The floating point result is automatically normalized and rounded unless otherwise specified (discussed later).

The operation does make use of the Q register (residue), so any vital information in Q prior to this instruction must be reloaded

2. Instruction: FLOATING MULTIPLY Mnemonic: FMU

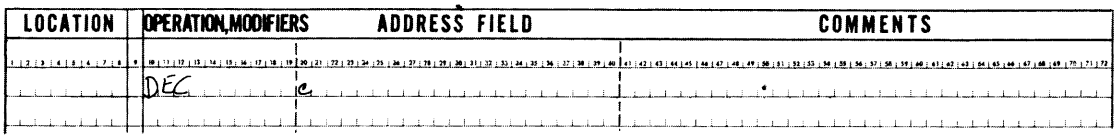
Elementary Form:



This instruction forms the product of two 48-bit quantities which are packed in floating point format. An operand is read from storage location m and is multiplied by the previous contents of the A register. The floating point result is automatically normalized and rounded unless otherwise specified (discussed later).

The operation does make use of the Q register (residue), so any vital information in Q prior to this instruction must be reloaded.

3. Instruction: Floating Point Data Initialization Mnemonic: DEC
Elementary Form:



This instruction has been introduced previously, however, only decimal integers were used using integer format. Here we discuss the same instruction but using floating point numbers and hence, the numbers must be prestored in floating point format.

Since integer and floating point formats are in no way related, it is important to note the following rules:

1. Integer arithmetic instructions may not operate on floating point operands.

2. Floating point arithmetic instructions may not operate on integer operands.

To do so will yield incorrect results. Below is a table separating the integer arithmetic instructions and floating point arithmetic instructions.

Integer Arithmetic Instructions	Floating Point Arithmetic Instructions
Add - ADD	Floating Add - FAD
Subtract - SUB	Floating Subtract - FSB
Multiply Integer - MUI	Floating Multiply - FMU
Divide Integer - DVI	Floating Divide - FDV
Replace Add One - RAO	
Replace Subtract One - RSO	
Replace Add - RAD	
Replace Subtract - RSB	
Increase A - INA	

Some of each you are already familiar with.

To prestore floating point instructions the DEC instruction is necessary. The DEC instruction is a pseudo instruction that can be used to prestore operands in integer or floating point format. If the value, c, contains no decimal point, the number is stored in integer form. If a decimal point is contained in the number, the number is stored in floating point form. Consider the two cards:

```
1.    DEC   5
2.    DEC   5.
```

In the first case the prestored number is in integer format and would have to be operated on with an integer arithmetic instruction. In the second case the prestored number is in floating point arithmetic instruction. The two formats differ greatly. A detailed discussion of these formats is given in Vol. II.

Problem 12 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	EVAL	
	ENTRY	EVAL	
Z	BSS	1	
SAVE	BSS	1	
CØN1	DEC	18.5	
CØN2	DEC	3.2	
EVAL	BSS	1	
	LDA	CØN1	
	FMU	X	18.5X
	STA	SAVE	
	LDA	CØN2	
	FMU	Y	3.2Y
	FAD	SAVE	
	STA	Z	
	SLJ	EVAL	
	END		

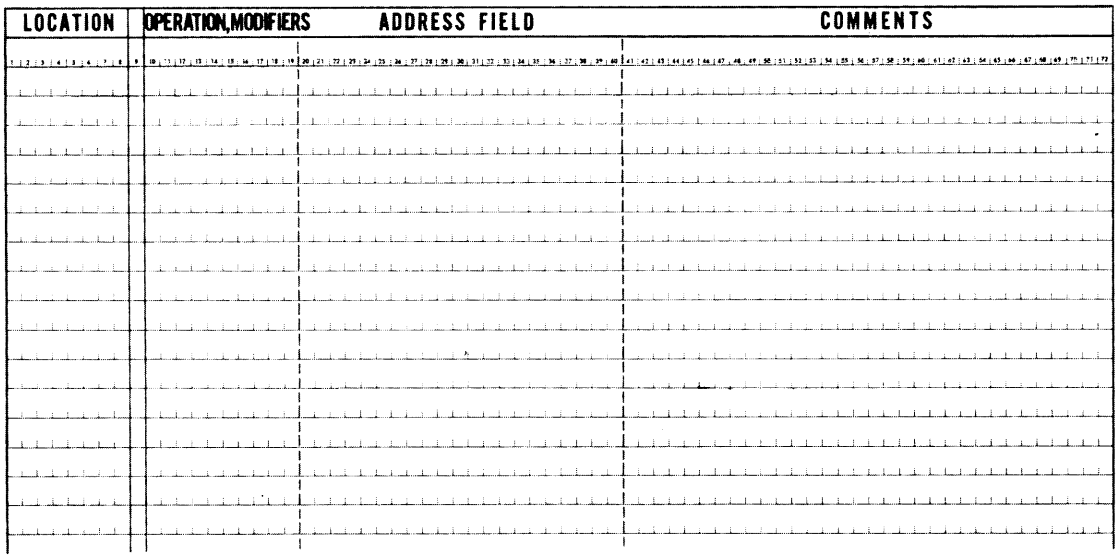
Somewhere within this subprogram would also be included the symbols X and Y in the location field with a declaration of the prestored data.

Student Problem 12A:

Evaluate $C = .5 \pi R^2 H - 150$ if R and H are given.

Flowchart:

Problem 12A could be solved by coding in the following manner:

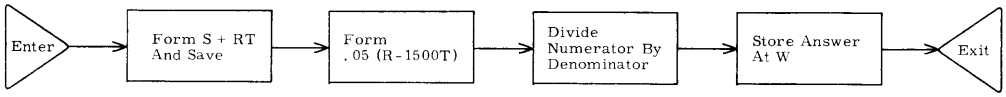


Problem 13:

Solve for W if R, S, and T are given.

$$W = \frac{.05\,(R-1500T)}{S+RT}$$

Flowchart:



The new instructions needed to solve this problem are:

1. FLOATING SUBTRACT - FSB
2. FLOATING DIVIDE - FDV

1. Instruction: FLOATING SUBTRACT Mnemonic: FSB

Elementary Form:

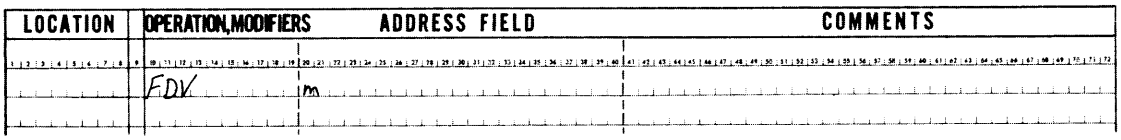
LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	FSB	m	

This instruction forms the difference of two 48-bit quantities which are both packed in floating point format. An operand is read from the storage location specified by m and is subtracted the previous contents of A. The floating point result is automatically normalized and rounded unless otherwise specified (discussed later).

The operation does make use of the Q register (residue), so any vital information in Q prior to this instruction must be reloaded.

2. Instruction: FLOATING DIVIDE Mnemonic: FDV

Elementary Form:



This instruction forms the quotient of two 48-bit quantities which are packed in floating point format. The contents of A are divided by an operand read from storage location m. The floating point result is automatically normalized and rounded unless otherwise specified (discussed later).

The operation does make use of the Q register (residue), so any vital information in Q prior to this instruction must be reloaded.

Division by zero constitutes a divide fault and represents an interruptible condition.

Problem 13 could be solved by coding in the following manner:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	FØRMULA	
	ENTRY	FØRMULA	
SAV1	BSS	1	
SAV2	BSS	1	
CØN1	DEC	1500.	
CØN2	DEC	.05	
W	BSS	1	
FØRMULA	BSS	1	
	LDA	R	
	FMU	T	
	FAD	S	
	STA	SAV1	= S + RT
	LDA	T	
	FMU	CØN1	
	STA	SAV2	= 1500T
	LDA	R	
	FSB	SAV2	
	FMU	CØN2	= .05(R-1500T)
	FDV	SAV1	
	STA	W	ANSWER
	SLJ	FØRMULA	
	END		

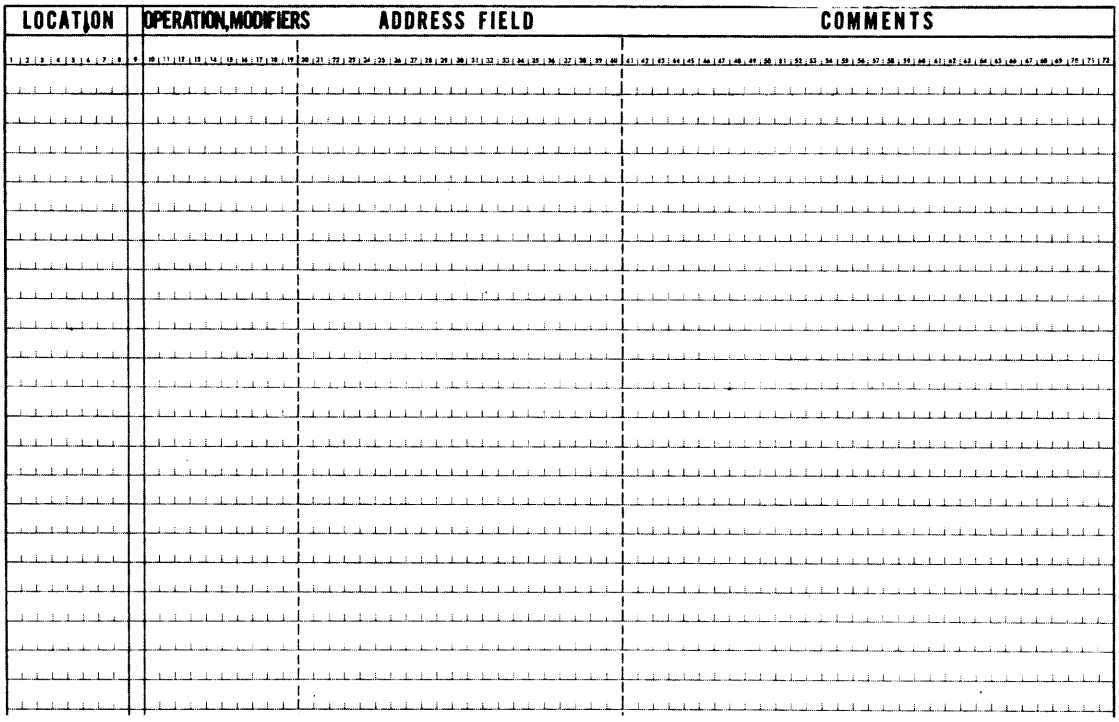
Somewhere within this subprogram would also be included the symbols R, S, and T in the location field with a declaration of the prestored data.

Student Problem 13A

Evaluate $$Z = \frac{15X^2 + 8.5XY - 10Y^2}{X - 10Y}$$

Flowchart:

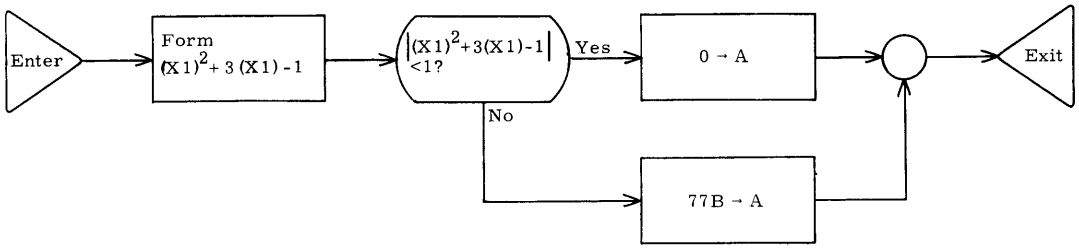
Problem 13A could be solved by coding in the following manner:



Problem 14:

Given the value X1 to be substituted into the function $X^2 + 3X - 1$, determine if the absolute value of the function is less than 1. If it is, exit with zero in A. If not, exit with octal 77 in A.

Flowchart:



The new instruction needed to solve this problem is:

1. LOAD A, MAGNITUDE - LDA, MG

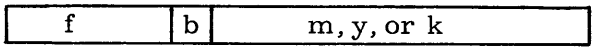
1. Instruction: LOAD A, MAGNITUDE Mnemonic: LDA, CM

Form:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	LDA,CM	m	

This instruction, even though it takes up one line of coding, actually consists of two hardware instructions and takes up 48 bits. A new concept is introduced called Augmenting and now we will take a little time to explain it.

Up to this point we have discussed several 24-bit instructions. Each instruction had the form:



The Compass format was one of the following:

1.	f	m, b
2.	f	y, b
3.	f	k, b

However, if you look at a reference manual or a code book, you will notice that the complete forms for 24-bit instructions entails more. For example:

LDA m, b

can be expanded to:

LDA, CM, MG (a)m, b, v

The modifiers CM and MG are operation field modifiers that change to a slight degree, the operation of the instruction. CM means, "load A with the complement of the operand". MG means "load A with the magnitude of the operand (absolute value). Either of these modifiers could be used.

The a within parentheses represents the bank that is to be referenced for the operand. Only the 3600 and 3800 systems have this feature. Besides the 15-bit address, the programmer can designate a specific bank. This would be if his program occupied more than one bank. He would have to be careful that he referenced not only the correct 15-bit address, but also the correct bank. If his program is completely contained in one bank, there is no need for designating bank terms, as the present program bank is automatically assumed.

The v modifier in the address field is another index designator and allows for increased address modification. At the beginning of this section we mentioned that the modified address $M = m+(B^b) + (V^v)$. V^v represents the same index registers that B^b represents.

For most 24-bit instructions inserting a modifier, a bank term, or a second index designator requires a hardware 24-bit augment instruction to be placed as the upper instruction of a memory word. The instruction itself then forms the lower part of the memory word. Together they form two distinct instructions, the upper augmenting the lower, and both taking up 48 bits.

Problem 14 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	FUNCTION	
	ENTRY	FUNCTION	
CON1	DEC	3.	
CON2	DEC	1.	
SAVE	BSS	1	
FUNCTION	BSS	1	
	LDA	X1	
	FMU	CON1	
	STA	SAVE	= 3*X1
	LDA	X1	
	FMU	X1	
	FAD	SAVE	
	FSB	CON2	
	STA	SAVE	= (X1)**2+3*X1-1
	LDA,MG	SAVE	
	FSB	CON2	
	AJP,PL	GREATER	
LESS	ENA	0	
	SLJ	FUNCTION	
GREATER	ENA	77B	77 OCTAL
	SLJ	FUNCTION	
	END		

Somewhere within this subprogram would also be included the symbol X1 in the location field with a declaration of the prestored data.

Student Problem 14A:

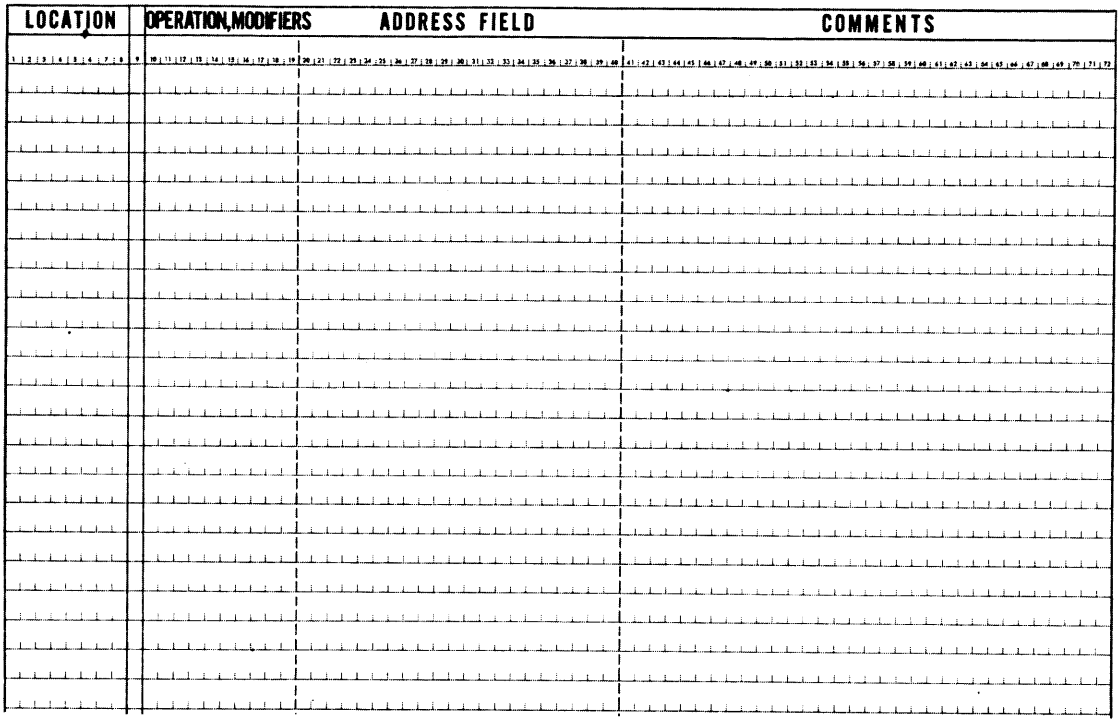
Given $Y = X^3 + 5.3X - 30$, a value is to be tested for X such that, when substituted in the function, $|Y|$ (absolute value) $< .001$

Substitute X and if,

1. $|Y| < .001$, enter 1 in A
2. $|Y| \geq .001$, enter 0 in A

Flowchart:

Problem 14A could be solved by coding in the following manner:



Problem 15:

Assume the following 9 x 9 array:

A11	A12	A13	A14	A15	A16	A17	A18	A19
A21	A22	A23	A24	A25	A26	A27	A28	A29
A31	A32	A33	A34	A35	A36	A37	A38	A39
A41	A42	A43	A44	A45	A46	A47	A48	A49
A51	A52	A53	A54	A55	A56	A57	A58	A59
A61	A62	A63	A64	A65	A66	A67	A68	A69
A71	A72	A73	A74	A75	A76	A77	A78	A79
A81	A82	A83	A84	A85	A86	A87	A88	A89
A91	A92	A93	A94	A95	A96	A97	A98	A99

This 81 element array is in memory starting at address ARRAY and is formed column by column as such:

1. A11
2. A21
3. A31
4. A41
5. A51
6. A61
7. A71
8. A81
9. A91
10. A12
11. A22

-
-
-

80. A89
81. A99

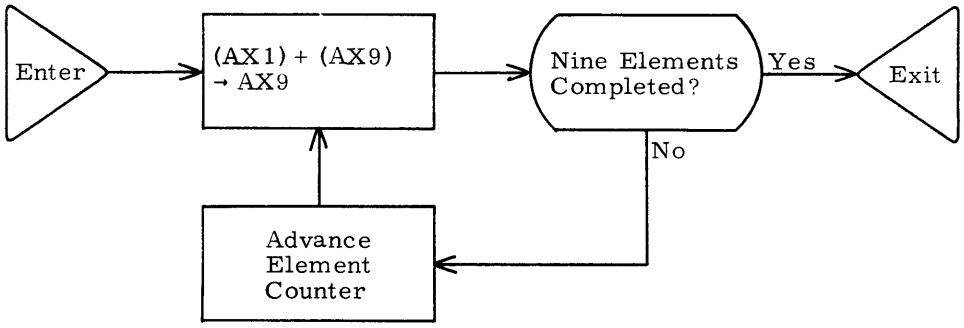
Write a subprogram that will add each element of the first column to its corresponding element of the last column and place each result at each address of the last column. The operation should go like this:

1. (A11)+(A19) →A19
2. (A21)+(A29) → A29

-
-
-

9. (A91)+(A99) → A99

Flowchart:



The new concept needed to solve this problem is the concept of Double Indexing. We mentioned before that most 24-bit instructions could be modified by:

1. operation field modifiers
2. bank terms
3. a second index designator

The second index designator allows double indexing. An instruction like the following is an example:

```
LDA          N, 1, 2
```

The address referenced is M where $M = m + (B^b) + (V^v)$. In this case $M = m + (B^1) + (V^2)$. Designators b and v represent the same six index registers, so if one knows how to interpret b, then he knows how to interpret v.

Problem 15 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	MATRIXMU	
	ENTRY	MATRIXMU	
MATRIXMU	BSS	1	
	ENI	0,1	1ST EL. ØF FIRST CØL.
	ENI	72,2	1ST EL. ØF LAST CØL.
A	LDA	ARRAY,2	
	FAD,RP	ARRAY,1,2	(AX1)+(AX9) AX9
	ISK	8,1	9 ELEMENTS?
	SLJ	A	NØ
	SLJ	MATRIXMU	YES, RETURN
	END		

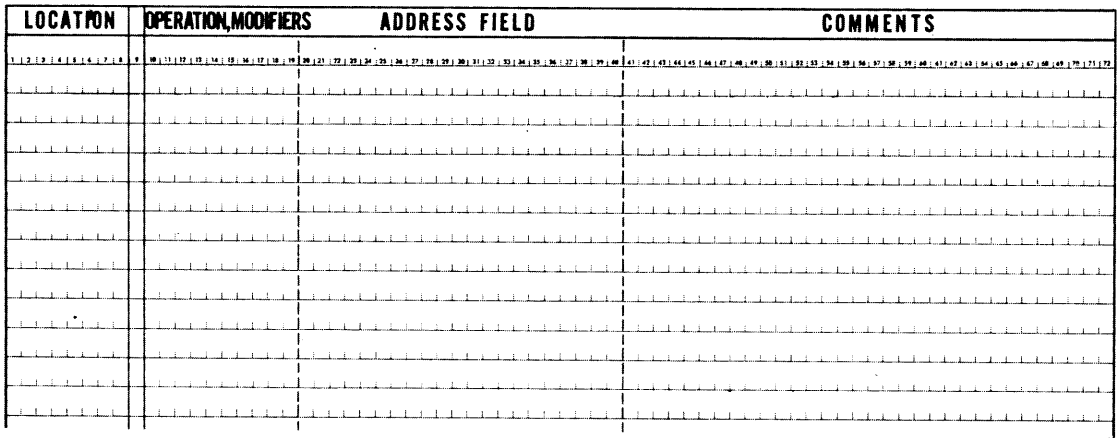
Somewhere within this subprogram would also be included the symbol ARRAY in the location field with a declaration of the prestored data or area reserved.

Student Problem 15A:

With respect to the preceding array subtract the 7th column elements from the middle column elements and store each result at the addresses of the 7th column.

Flowchart:

Problem 15A could be solved by coding in the following manner:



Problem 16:

A random set of 100 integers ranging from 1-100 and starting at address SCORES is to be separated into groups according to the following pattern:

1. scores less than 70 stored beginning at address UNSAT

2. scores greater than or equal to 70 but less than 76 stored beginning at address POOR

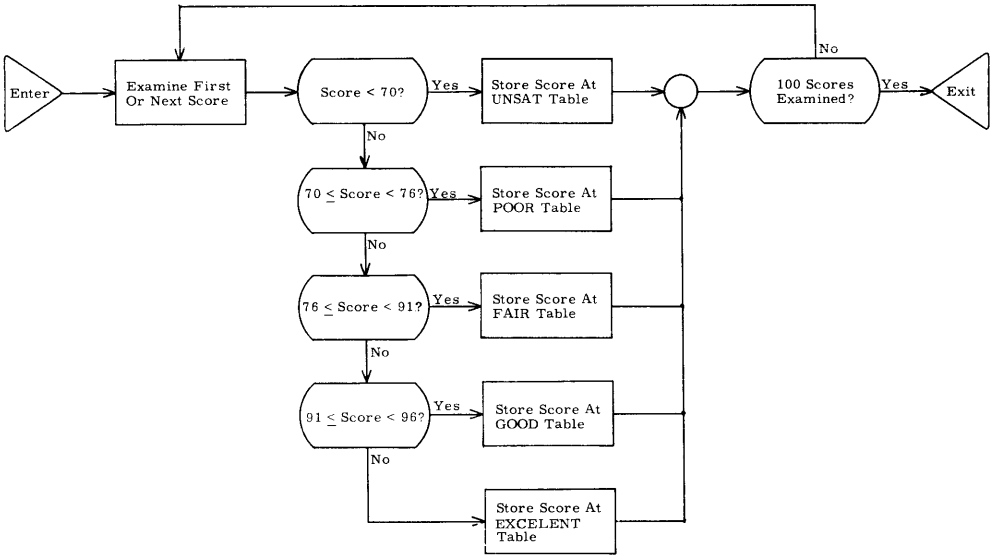
3. scores greater than or equal to 76 but less than 91 stored beginning at address FAIR

4. scores equal to or greater than 91 but less than 96 stored beginning at address GOOD

5. scores equal to or greater than 96 stored beginning at address EXCELENT

Write a subprogram that will compare the integers and store them accordingly.

Flowchart:



The new instruction needed to solve this problem is:

1. REGISTER JUMP - RGJP

1. Instruction: REGISTER JUMP Mnemonic: RGJP

Form:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	RGJP,s	p,y,m,b	

This instruction is the first 48-bit instruction encountered. It takes up 48 bits whether modifiers are specified or not. The 48-bit instructions normally have four characters for their Compass operation code.

This instruction is basically a compare instruction. It can compare any operational register with the value y, which is a 15 bit quantity. This is a fast

instruction since no memory reference is made.

The instruction compares the value in the register specified by p with the value y. The Compass mnemonics for each register can be found in the 3600 Instant Compass booklet, page 8 (pub. no. 60056500). The basic comparisons are:

1. (p) = y
2. (p) > y
3. (p) < y
4. (p) ≠ y
5. (p) ≤ y
6. (p) ≥ y

The particular comparison is specified by the operation modifier, s. For each of the above:

1. s = EQ
2. s = GT
3. s = LT
4. s = NE
5. s = LE
6. s = GE

If the two quantities compare as tested (yes answer), a jump is made to address M where M = m + (B^b). If the two quantities do not compare as tested (no answer), a full exit is taken and the program continues.

Problem 16 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	SEPARATE	
	ENTRY	SEPARATE	
UNSAT	BSS	100	
POOR	BSS	100	
FAIR	BSS	100	
GOOD	BSS	100	
EXCELENT	BSS	100	
SEPARATE	BSS	1	
	ENI	0,1	
	ENI	0,2	
	ENI	0,3	
	ENI	0,4	
	ENI	0,5	
	ENI	0,6	
NEXT	LDA	SCORES,6	
	RGJP,LT	A,70,LT70	(A) LT70
	RGJP,LT	A,76,LT76	(A) LT76
	RGJP,LT	A,91,LT91	(A) LT91
	RGJP,LT	A,96,LT96	(A) LT96
GE96	STA	EXCELENT,5	
	INI	1,5	
TEST	ISK	99,6	100 SCORES TESTED?
	SLJ	NEXT	NO
	SLJ	SEPARATE	YES
LT70	STA	UNSAT,1	
	INI	1,1	
	SLJ	TEST	
LT76	STA	POOR,2	
	INI	1,2	
	SLJ	TEST	
LT91	STA	FAIR,3	
	INI	1,3	
	SLJ	TEST	
LT96	STA	GOOD,4	
	INI	1,4	
	SLJ	TEST	
	END		

Somewhere within this subprogram would also be included the symbol SCORES in the location field with a declaration of the prestored data or the area reserved.

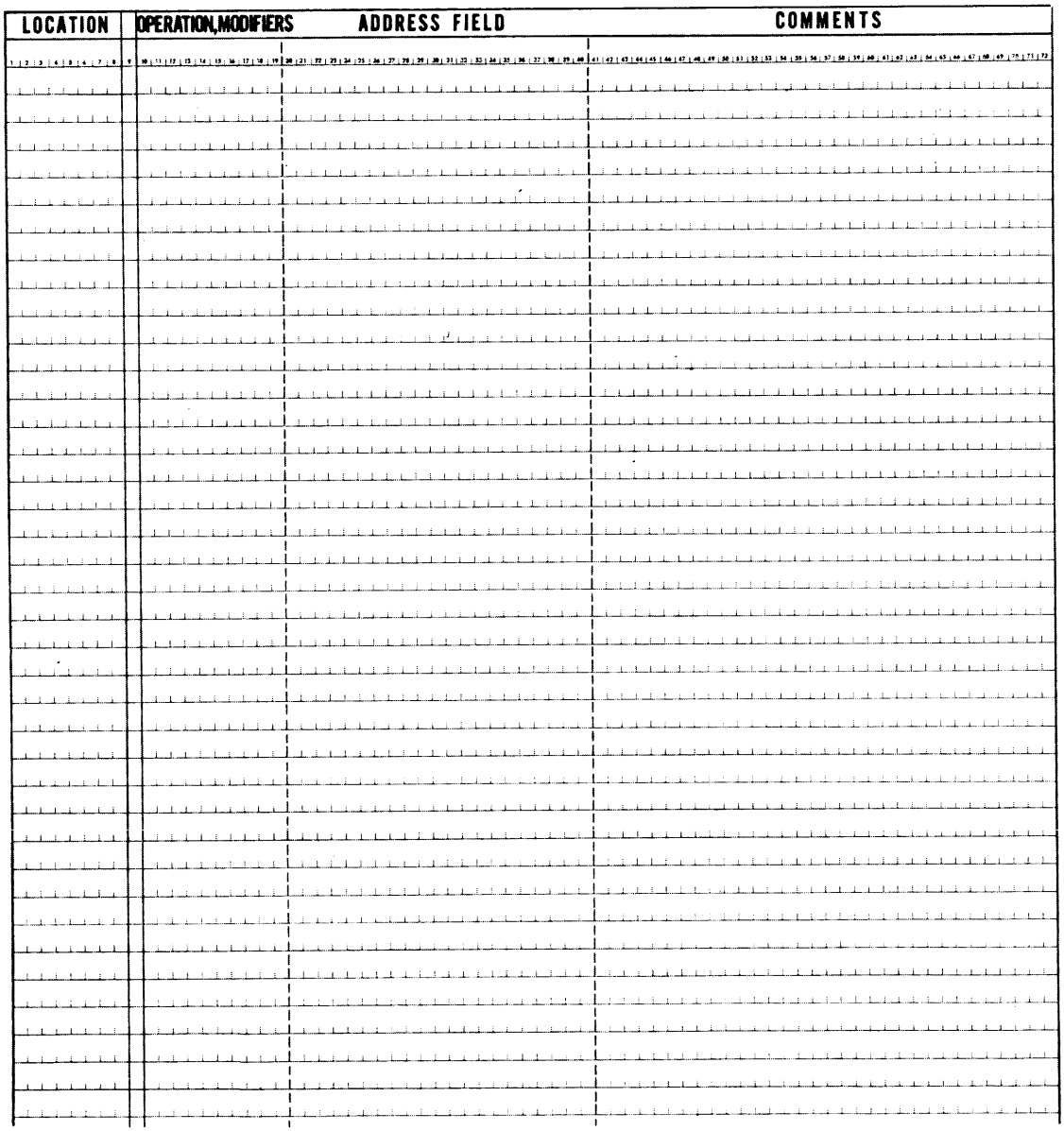
Student Problem 16A:

Given the function $Y = X^2 + 5X - 3$.

Assume there exists in core 100 integer values starting at address X to be substituted one at a time for X. If any of the substitutions causes Y to be between -5 and +5 (inclusive), store that value for X in a table starting at address ROOT.

Flowchart:

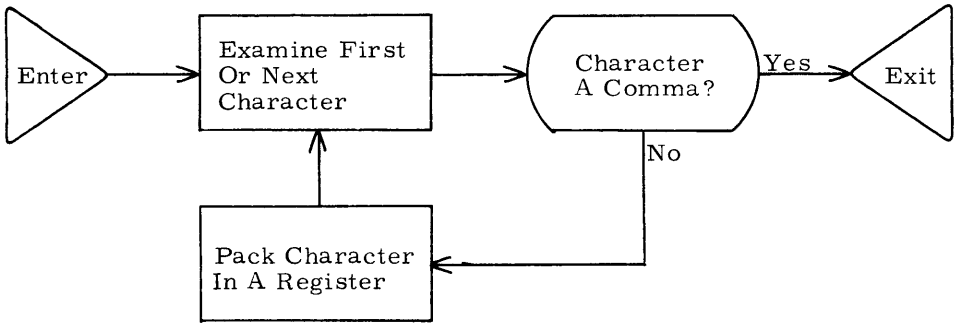
Problem 16A could be solved by coding in the following manner:



Problem 17:

Assume an 80 column card has been read into memory in BCD format starting at address CARD, each column forming a 6 bit internal BCD character. Assuming the word before the first comma on the card to be less than or equal to 8 characters, write a program to load the word into the A register. The internal BCD equivalent to a comma is 73B.

Flowchart:



The new instruction needed to solve this problem is:

1. LOAD BYTE - LBYT

1. Instruction: LOAD BYTE Mnemonic: LBYT

Form:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	LBYT, Ao, Ee, LI, CL m, b, v		
	Qo RI		

This instruction is an instruction that transmits a "byte" of a memory word from a memory location to the A or Q register, which ever is specified. In effect this instruction is like the LOAD A or the LOAD Q. The only difference is the variable size byte from any portion of the memory word can be transferred to any part of the A or Q register.

Since there are so many modifiers, let us begin by defining a few.

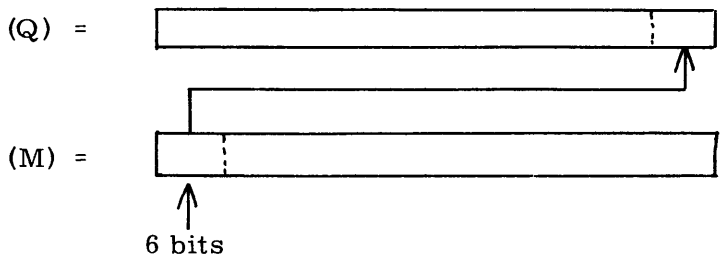
1. Ao or Qo - One of these must be specified. This represents the off-set designator of the A or Q register. The off-set designator is the bit position (0-47) representing the rightmost bit of the byte in the register.

2. Ee - This modifier represents the size of the byte that is to be transferred (1-47).

3. (V^v) - The contents of index register v represent the off-set designator is the bit position (0-47). The off-set designator is the bit position (0-47) representing the rightmost bit of the byte in the memory word. The off-set designator must be entered in V^v prior to this instruction.

4. $M = m + (B^b)$ - This represents the memory location referenced.

This information gives enough to transmit a byte. Suppose you wanted to transmit the uppermost 6 bit byte to the lowest part of the Q register, let's say from address M. How would you code it?



Two instructions are needed and would look like the following:

```
ENI                 42, 1
LBYT, Q0, E6          M, , 1
```

Another modifier can be introduced at this time.

CL - Means clear the complete destination as the byte is being transferred.

In this case it concerns the Q register. If CL is not specified, the rest of Q will be untouched. If CL is specified, the rest of Q will be zeroed out.

As long as LI or RI are not specified, this instruction will always full exit when completed.

If either LI or RI is specified, the complexity of the situation changes, and a new concept for this instruction is introduced - INDEXING. Indexing means that the memory off-set designator can act as a pointer and can move to the right (right indexing) or left (left indexing) each time the instruction is executed. This means that sequential bytes can be loaded and examined and the pointer moves automatically. A check is also made, that if the memory pointer completes a word, it will automatically provide for re-establishing the pointer at the next word. After the loading of the byte has taken place, a check is made to see if another byte could be taken from the present memory word. If yes, a skip exit (P + 2) is taken. If no, a full exit (P + 1) is taken. The programmer has a maximum of two instructions to re-establish the pointer (contents of V^V) and increment (or decrement) the memory address referenced (B^b).

Let's return to problem 17 and see how the word before the first comma could be loaded into the A register.

Problem 17 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	UNPACK	
	ENTRY	UNPACK	
UNPACK	BSS	1	
	ENI	0,1	
	ENI	42,2	MEM. OFFSET DESIGNATOR
A	LBYT,QQ,E6,RI,CL CARD,1,2		
	ENI	42,2	EXECUTED ONLY WHEN
	INI	1,1	COMPLETE WORD EXAMINED
	RGJP,EQ	Q,73B,UNPACK	SKIP EXIT TO HERE
	QLS	42	
	LLS	6	CHAR TO A
	SLJ	A	NEXT CHAR
	END		

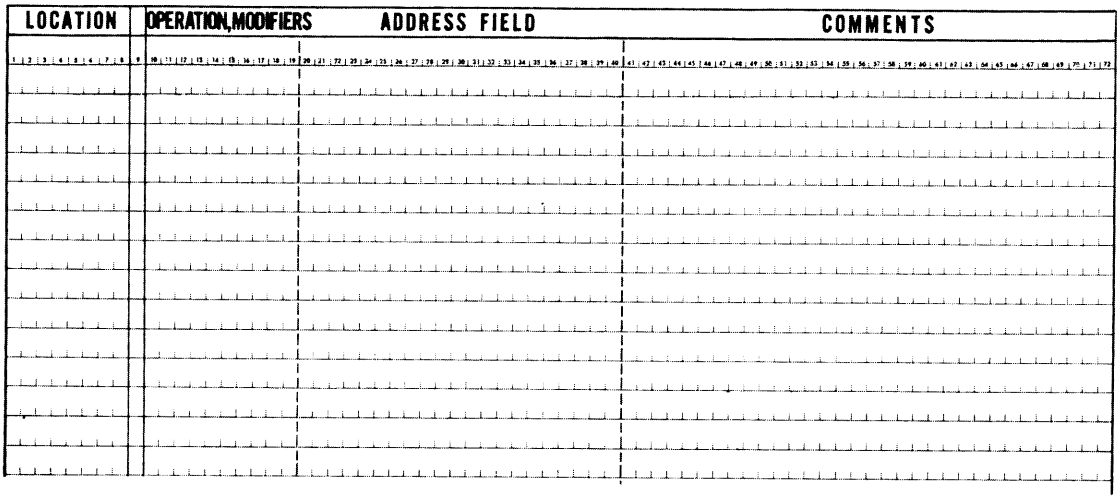
Somewhere within this subprogram would also be included the symbol CARD in the location field with a declaration of the prestored data or the area reserved.

Student Problem 17A:

Assume an 80 column internal BCD card image in memory starting at address CARD. Write a subprogram that will load A with the characters from column 10 up to but not including the first blank encountered. The internal BCD equivalent to a blank is 60B.

Flowchart:

Problem 17A could be solved by coding in the following manner:



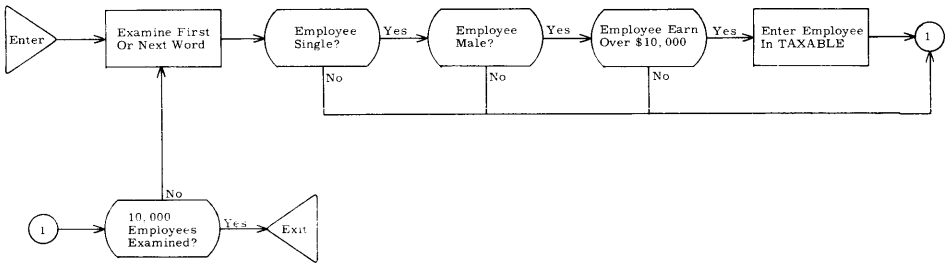
Problem 18:

A company employs 10,000 people. For each employee information is contained in a 48-bit word concerning his status. The uppermost three bits mean the following:

Bit 47:	1 = salary over $10,000
	0 = salary less than or equal to $10,000
Bit 46:	1 = Male
	0 = Female
Bit 45:	1 = Married
	0 = Single

If each employee's status is contained within a table in memory starting at address EMPLOY, transmit all words of employees who are single men and earn over $10,000 to a table starting at address TAXABLE.

Flowchart:

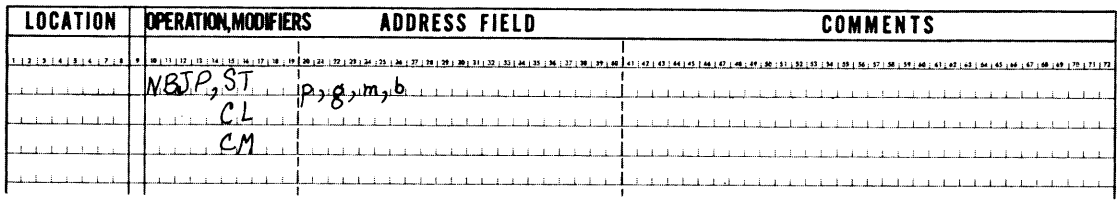


The new instruction needed to solve this problem is:

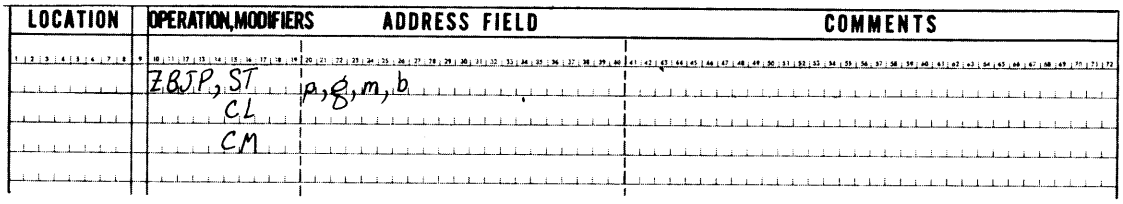
1. BIT SENSING - NBJP
   ZBJP

1. Instruction: BIT SENSING Mnemonic: NBJP
ZBJP

Form:



Or Form:



The BIT SENSING instruction has two forms because one checks a bit for being non-zero (1) and the other checks a bit for being zero. Both instructions are fast since neither of them requires a memory reference.

This instruction checks any bit in any operational register for zero or non-zero. The register is specified by p and the bit address of the register is specified by g. The designator g can range from 0 through 47. For the NBJP instruction, if the bit is a "1", a jump is made to address M where $M = m + (B^b)$. If it is an "0", a full exit is taken.

For the ZBJP instruction, if the bit is an "0", a jump is made to address M. If it is a "1", a full exit is taken.

After the bit is checked and the exit is determined, the programmer has the option to change the bit or leave it as it was. The following forms are allowed for the operation field (assume NBJP):

	INSTRUCTION	ACTION
1.	NBJP	BIT IS LEFT UNTOUCHED
2.	NBJP, ST	BIT IS THEN SET
3.	NBJP, CL	BIT IS THEN CLEARED
4.	NBJP, CM	BIT IS THEN COMPLEMENTED

Problem 18 could be solved by coding in the following manner:

LOCATION	OPERATION, MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	TAX	
	ENTRY	TAX	
TAXABLE	BSS	10000	
TAX	BSS	1	
	ENI	0,1	
	ENI	0,2	
NEXT	LDA	EMPLØY,1	
	NBJP	A,47,A	
LAST	ISK	9999,1	LAST EMPLOYEE
	SLJ	NEXT	NØ
	SLJ	TAX	YES
A	NBJP	A,46,B	
	SLJ	LAST	
B	ZBJP	A,45,ENTER	FØUND ØNE
	SLJ	LAST	
ENTER	STA	TAXABLE,2	
	ENI	1,2	
	SLJ	LAST	
	END		

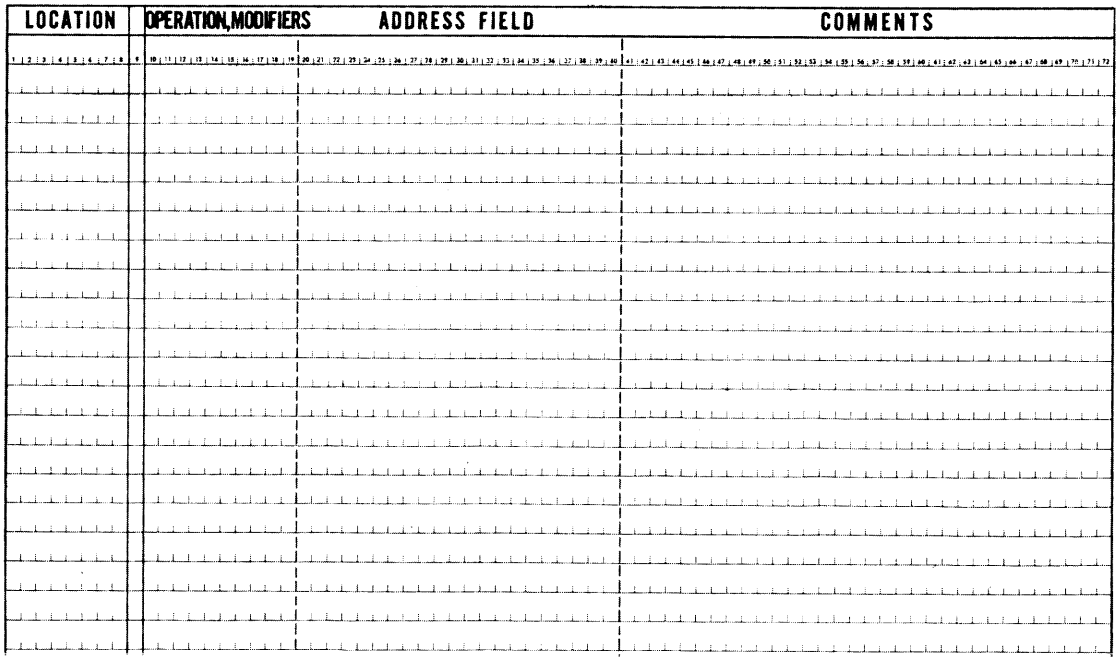
Somewhere within this subprogram would also be included the symbol EMPLOY in the location field with a declaration of the prestored data on the area reserved.

Student Problem 18A:

With respect to the previous problem, write a subprogram that will store the status of all married women making over $10,000. Start this table at address JOINTAX.

Flowchart:

Problem 18A could be solved by coding in the following manner:

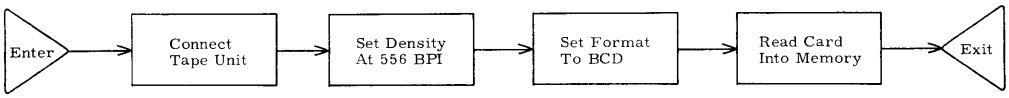


Problem 19:

Magnetic tape unit #3 is physically connected to data channel #0, controller #2. If the next record on tape is a card image of 80 BCD characters recorded at 556BPI, read the card into memory starting at address CARD.

If the equipment cannot be referenced for any reason, enter A with 77 octal and exit subprogram.

Flowchart:



The new instructions needed to solve this problem are:

1. CONNECT - CONN
2. EXTERNAL FUNCTION - EXTF
3. BEGIN READ - BEGR

1. Instruction: CONNECT Mnemonic: CONN

Form:

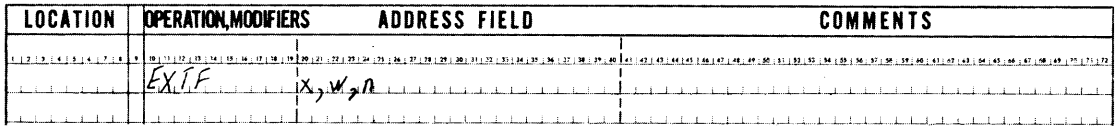
LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	CØNN	x,e,u,n	

This instruction connects one equipment and one unit of that equipment to data channel x. The data channel is specified by numbers $0\text{-}31_{10}$ for the 3600/3800 systems and 0-3 for the 3400 system since these are the maximum number of data channels allowed.

The e and u designators specify the equipment and unit numbers, respectively. The range for e is 0-7 since a maximum of 8 controllers are allowed. The range for u is 0 up to the maximum units the controller can feed. The maximum to date is 16_{10} (magnetic tape). For each equipment (controller) and unit there is a switch or a dial on it that can be manually changed, so that it can be referenced if its number is specified in the program.

If the connection can take place, it does so immediately, and a full exit occurs. If for some reason the connection does not take place (power down, unmatched numbers, etc.), a jump is made to the address specified by the designator n.

2. Instruction: EXTERNAL FUNCTION Mnemonic: EXTF
Form:

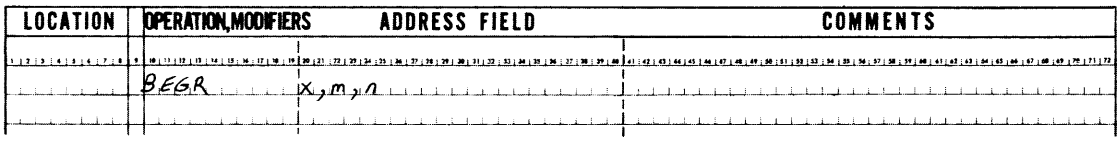


This instruction transmits a 12 bit function code w to the unit connected to data channel x. The function codes for some of the units can be found in the 3000 Series Peripheral Equipment Reference Manual, pub. no. 60108800.

If the unit interprets and can perform the specified function, it initiates it and the instruction immediately full exits. If for some reason the function cannot be performed (channel busy, illegal code, etc.), a jump is made to the address specified by n.

3. Instruction: BEGIN READ Mnemonic: BEGR

Form:



This instruction initiates the transmission of data from external equipment to memory. The equipment referenced is that connected to data channel x.

Two things are lacking from this instruction; namely, the number of 48-bit words to be transferred and the starting address in memory.

This is too much information to be contained in the instruction. Associated with this instruction is a control Word Address (a)m at which is prestored prior to this instruction the Word Count and the Starting Address. How does the programmer prestore this information at the Control Word Address? In order to explain this let us generally examine the general format of the Control Word and see what more it contains.

There are four basic formats used to prestore Control Words:

1.	IOSW, C	(a)m, w
2.	IOTW, C	(a)m, w
3.	IOSR, C	(a)m, w
4.	IOTR, C	(a)m, w

Each format represents a 48-bit word. The designator (a)m represents the Starting Address (bank term not necessary) and the designator w represents the Word Count, i.e., the number of 48-bit words to be transferred. Why, then, are there different operation codes and what is the modifier C? These are other features built into the system.

First of all, what is the difference between IOSW, IOTW, IOSR, and IOTR? Here are the definitions:

1. IOSW - Do the IO but skip w words. This means that w words are read but not transmitted. In this way the programmer can skip information.

2. IOTW - Do the IO and actually transmit w words to memory. This means that words are continually read until the w count is satisfied, even if it means reading multi-records or multi-files.

3. IOSR - Do the IO but skip to end-of-record. This means that a record can be skipped, even though the programmer does not know its length. Care should be taken to specify a large word count, enough to cover the skipped record. (See IOTR).

4. IOTR - Do the IO and actually transmit words until either an end-of-record is sensed or the w count is satisfied. This means that a programmer can read a record of any size and then have it terminate. Care should be taken that the w count is sufficiently high to insure the reading of the complete record. If the w count is smaller than the number of 48-bit words in the record, some of the words will not be transmitted.

Secondly, what is the modifier C? The modifier C is optional and specifies "chaining". Chaining means that when that control word is finished, either by end-of-record being sensed or by the w count being satisfied, that another control word is picked up (at Control Word Address +1) and the operation continues. There is no waiting or stopping of the peripheral equipment when this is done.

To illustrate this concept suppose a programmer wished to bypass the first record on tape and read in the second. Here's an example of the control words needed:

```
CWA1        IOSR,C          0,5000
            IOTR            FWA1.5000
```

Both control words will go until end-of-record is sensed or until the w count is satisfied (we assume records of less than 5000 words). A ficticious first word address or zero is used in the first control word since no words go to memory and to prevent an assembler diagnostic. The first control word after skipping the first record "chains" into the control word. The second control word immediately transmits the second record to memory starting at address FWA1. When this control word is finished, the read operation terminates since chaining is not further specified. Now let's return to problem 19 and see how we might solve it.

Problem 19 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	INPUT	
	ENTRY	INPUT	
CARD	BSS	10	
CWA1	IATW	CARD,10	8 CHAR PER WØRD
REJ	ENA	7,7B	
INPUT	BSS	1	
	CØNN	0,2,3,REJ	
	EXTF	0,3,REJ	SS6BPI
	EXTF	0,2,REJ	BCD FØRMAT
	BEGR	0,CWA1,REJ	
	SLJ	INPUT	
	END		

Problem 19A:

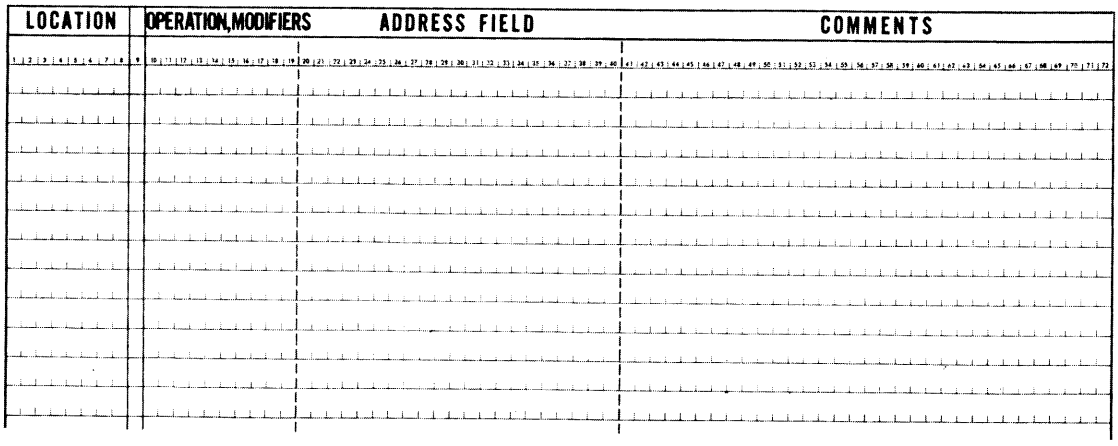
Write a subprogram that will:

1. connect channel #1 to equipment #2 unit #3
2. set density to 800 (assume code is 6)
3. set format to binary (assume code is 1)
4. bypass the first two records
5. read the third record to memory starting at address REC3.

If the equipment cannot be referenced for any reason, exit subprogram with (A) = 7.

Flowchart:

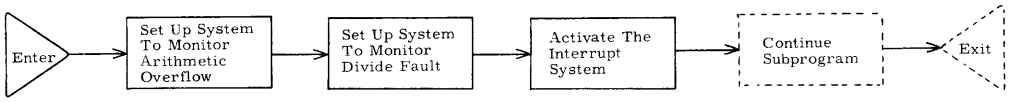
Problem 19A could be solved by coding in the following manner:



Problem 20:

Assume a particular routine contains many fixed point add, subtract, multiply and divide instructions. In order to monitor any faults of Arithmetic Overflow or Divide Fault the routine requires the interrupt system to be set up. Write the entrance to the subprogram such that the computer would monitor either of these faults for the remainder of the subprogram.

Flowchart:



This problem deals with hardware interrupt and is used to acquaint the reader with the 3600/3800 interrupt system. It must be noted that at times instructions used to solve this problem are illegal. The Scope monitor, before giving control to the Compass subprogram, will set up the interrupt system so as to make any attempt to change interrupts illegal. This problem is designed for the systems programmer rather than the applications programmer.

At this time let us say a few words about the upper 3000 interrupt system in general.

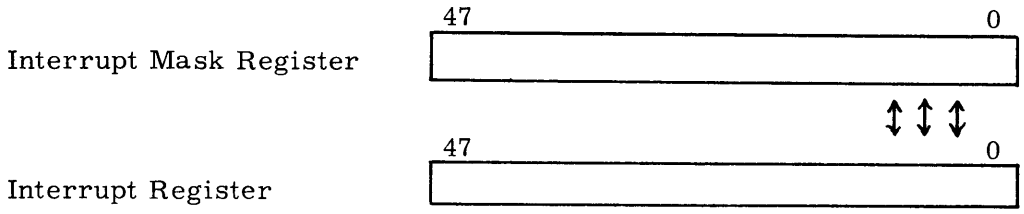
Within the system there are two registers that specifically deal with interrupt. They are:

1. Interrupt Mask Register
2. Interrupt Register

The Interrupt Mask register is a register the bits of which represent each type of possible interrupt. These bits can be set or cleared under program control (An exception is the bits representing the data channels in the 3600/3800 systems. They are hardware set and cannot be cleared.) A particular bit is set by the programmer if he wishes to monitor a particular function and have the computer interrupt if that

function arises. If the mask bit is not set, and the function arises, no interrupt will take place and the program will continue. So we can say that the programmer can use the Interrupt Mask register to monitor any of the functions in order to interrupt his program if the function takes place.

The Interrupt register is closely associated with the Interrupt Mask register. The bits have a one-to-one correspondence so that the registers logically are pictured thus:



A bit is set in the Interrupt register when that particular function actually takes place and its mask bit was set. When the interrupt routine is entered, a "logical product" of the two registers is taken by the programmer. Any binary position that yields a "1" determines the cause of interrupt and the programmer can jump to the routine that takes care of that function.

We mentioned that before an interrupt can actually take place on a function, its mask bit must be set. There is one more condition that must be present. The interrupt system must be active. If the interrupt system is active and the functional mask bit is set, interrupt will take place when that function arises. If the interrupt system is not active, no interrupt will take place even though the mask bit is set.

In summary, then, these conditions must be present in order to force the computer into interrupt:

1. Interrupt system must be active.
2. The mask bit must be set for the particular function.
3. The function representing the mask bit actually arises.

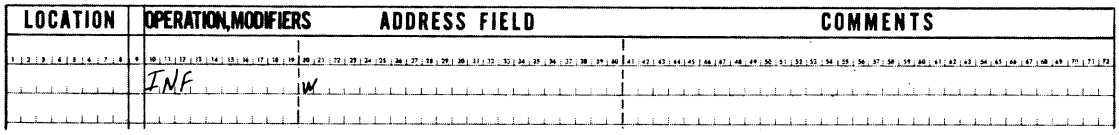
If any of these are lacking, interrupt will not take place and the main program will continue.

The new instruction needed to solve this problem is:

1. INTERNAL FUNCTION - INF

1. Instruction: INTERNAL FUNCTION Mnemonic: INF

Form:



This instruction is a general instruction that has many purposes, all based on the code w. The codes dealing with the interrupt system are the following:

w code	Function	Meaning
1	Clear Shift Fault Interrupt	These codes individually clear its bit in the Interrupt register. This is done after the program has determined that this bit has caused the interrupt. It discontinues the signal so that the computer is not re-interrupted and caught in a never-ending loop.
2	Clear Divide Fault Interrupt	
3	Clear Exponent Overflow Interrupt	
4	Clear Exponent Underflow Interrupt	
5	Clear Arithmetic Overflow Interrupt	
7	Clear Internal Reject Interrupt	
9	Clear Real Time Clock Interrupt	
10	Clear Storage Reference Fault Interrupt	
11	Clear 1604 Mode Interrupt	
12	Clear Trace Mode Interrupt	
13	Clear Bounds Interrupt	
14	Clear Illegal Instruction Interrupt	
15	Clear Operand Parity Error Interrupt	
16	Clear Manual Interrupt	
17	Clear All Internal Interrupts	Clears the entire Interrupt register.
18	Set Interrupt Active	Activates the interrupt system. Without the system active the interrupt signals will still be generated, but the computer will not be interrupted.
21	Clear Interrupt Active	Inactivates the interrupt system. In this state no interrupt signal will interrupt the computer.

The first few codes clear individual bits in the Interrupt register. This is done only after the source of interrupt has been determined by the programmer. He clears it out. The hardware will not. If the programmer does not clear it out, the interrupt routine will be re-entrant and form a never-ending loop.

The last two codes set the interrupt system active and clear it out. The system must be set active in order for any interrupts to occur.

Problem 20 could be solved by coding in the following manner:

LOCATION	OPERATION,MODIFIERS	ADDRESS FIELD	COMMENTS
	IDENT	ARITHFUN	
	ENTRY	ARITHFUN	
ARITHFUN	BSS	1	
	NBJP,ST	IM,1,*+1	SET DIV. FAU. BIT
	NBJP,ST	IM,4,*+1	SET ARITH. ØV. BIT
	INF	18	SET INT. ACT.
			CONT. PRØG.
	.		
	.		
	.		
	.		
	SLJ	ARITHFUN	RETURN
	END		

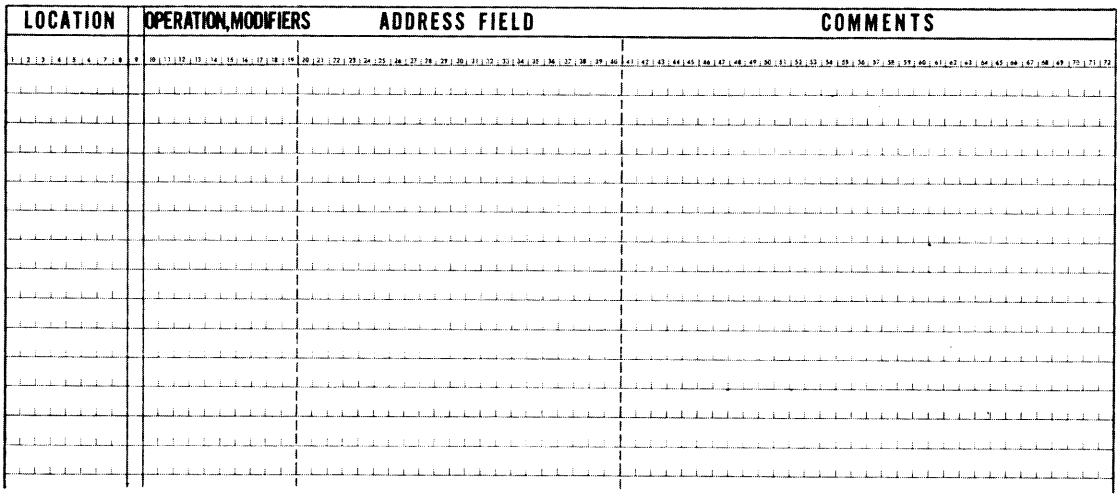
Student Problem 20A:

Write the entrance to a subprogram that will allow the program to interrupt on any of the following conditions:

1. Real Time Clock
2. Storage Reference Fault
3. Bounds Fault
4. Illegal Instructions Fault

Flowchart:

Problem 20A could be solved by coding in the following manner:



CONTROL DATA
CORPORATION

CORPORATE HEADQUARTERS, 8100 34th AVE. SO., MINNEAPOLIS, MINN, 55440
SALES OFFICES AND SERVICE CENTERS IN MAJOR CITIES THROUGHOUT THE WORLD

Pub. No. 60166700 Rev. B

Litho in U.S.A.