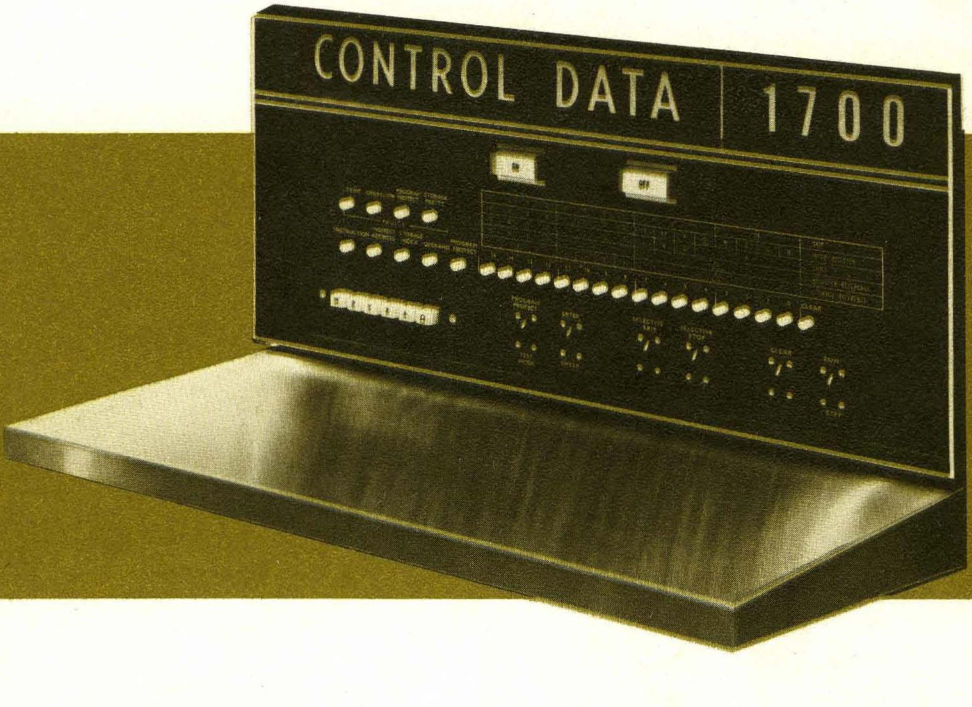
1700

COMPUTER MAINTENANCE

Volume I

SECOND EDITION



CONTROL DATA INSTITUTE

1700 COMPUTER

MAINTENANCE TRAINING MANUAL, VOLUME I

CONTROL DATA CORPORATION

For Training Purposes Only

This book was compiled and written by instructors of

Training Department
Control Data Institute
CONTROL DATA CORPORATION

Publications Number
60169500B
August, 1970

REVISION RECORD

REVISION	DESCRIPTION
(2-66)	First edition.
(7-67)	Manual revised.
A	
(8-10-70)	Manual revised. Affected pages are title page, revision record, 18, 23, 24, 25, 26, 27, 28, 29, 30
B	31, 1-1, 1-2, 1-4, 1-5, 1-10, 1-31, 2-2, 2-3, 2-7, 3-4, 3-13, A-1, A-2, and A-26.

Publication No.
60169500B

Address comments concerning this manual to:

CONTROL DATA EDUCATION INSTITUTES
4530 West 77th Street
Minneapolis, Minnesota 55435

ATTN: Advanced Technology
Development Division
Hardware Development, PGT310

FOREWORD

This manual has been prepared to acquaint the reader with the CONTROL DATA 1700 Computer instructions. Information covers the operation of all 1700 machine language instructions and addressing modes. The Appendix contains flow diagrams of most 1700 instructions and also includes various tables to facilitate the reading or writing of 1700 machine language programs.

Work projects are interspersed throughout the manual and should be completed before continuing to subsequent areas. Answers to the projects are given at the end of the chapter containing the project.

In any technical writing effort, possibilities of errors are always present. Although Control Data Institute makes a conscious effort to minimize errors in its publications, errors are nevertheless inevitable. If you would like to make the existence of errors known, or would like to make comments or suggestions concerning the manual, you might find the Comments Sheet at the end of the manual to be of help. Forward your comments to the Educational Development Section, Control Data Institute, 3255 Hennepin Avenue South, Minneapolis, Minnesota 55408.

GENERAL TABLE OF CONTENTS

CONTENTS

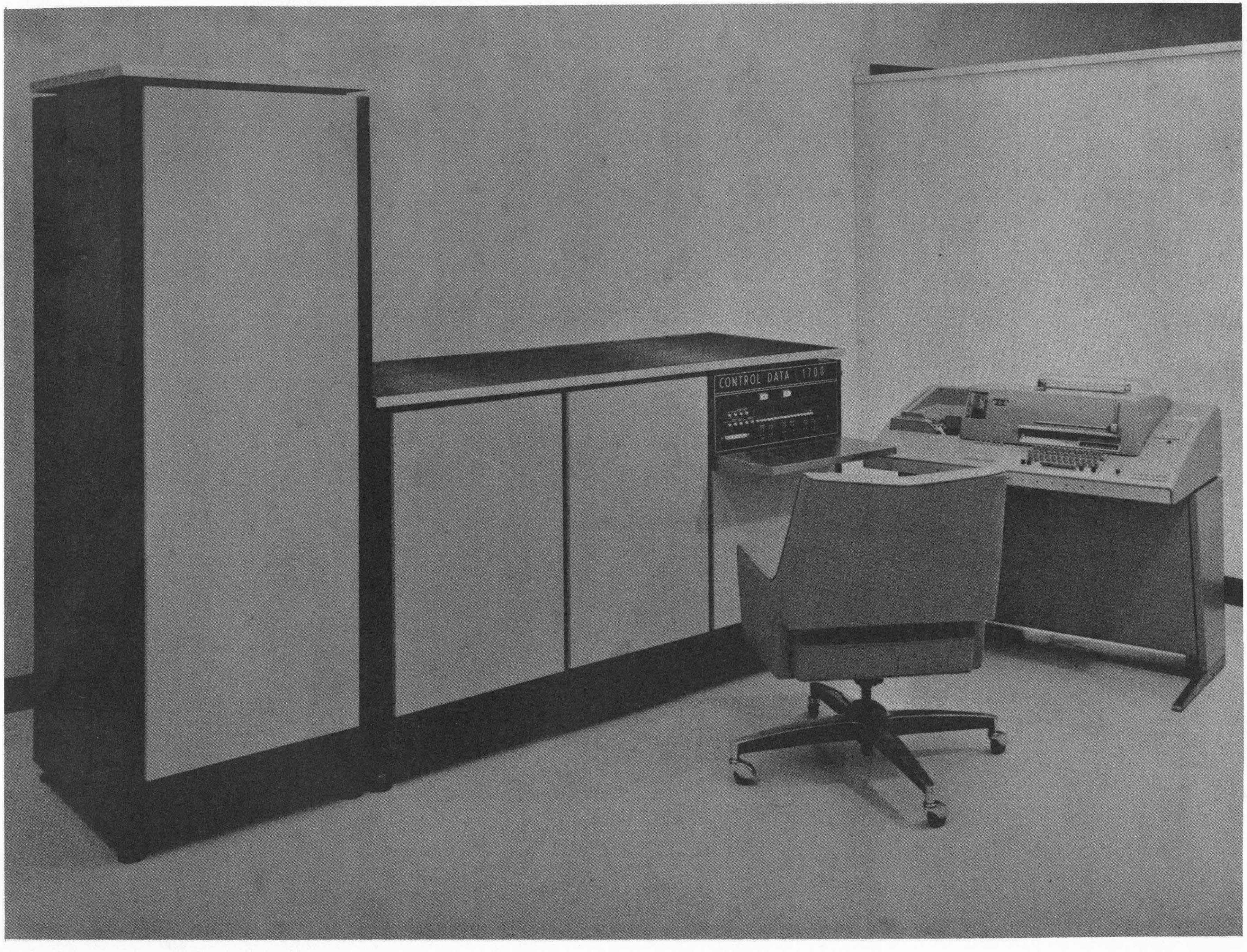
CHAPTER I NON-ADDRESSABLE INSTRUCTIONS

CHAPTER II ADDRESSABLE INSTRUCTIONS

CHAPTER III ADDRESSING MODES

APPENDIX FLOW DIAGRAMS & TABLES

INTRODUCTION TO THE 1700 COMPUTER SYSTEM


CONTROL DATA | 1700

INTRODUCTION TO THE 1700 COMPUTER SYSTEM

BASIC INDUSTRIAL CONTROL SYSTEM CONCEPTS

The world in which you live is an analog world. You continually receive analog representations of the physical phenomena from your environment. The other animal and plant life of the environment also receive and respond to the same analog representations. Man, however, in his development has found it necessary to convert these analog representations into a form in which they can be more easily recognized and measured.

Analog representations appear in many forms; for example, a temperature, the flow of fluid, the flow of electricity, atmosphere or other gas pressures and complex chemical compositions.

To become very basic in our early discussion of the analog world, let's look at some well-known examples. The thermostat in your home is a good example of man's conversion of analog representations into a recognizable and measurable form. The thermostat on your wall is normally a bi-metal instrument; that is, an instrument that has as its heart two pieces of metal bonded together. The two metals expand at different rates so that as the temperature changes this bonded unit will tend to warp.

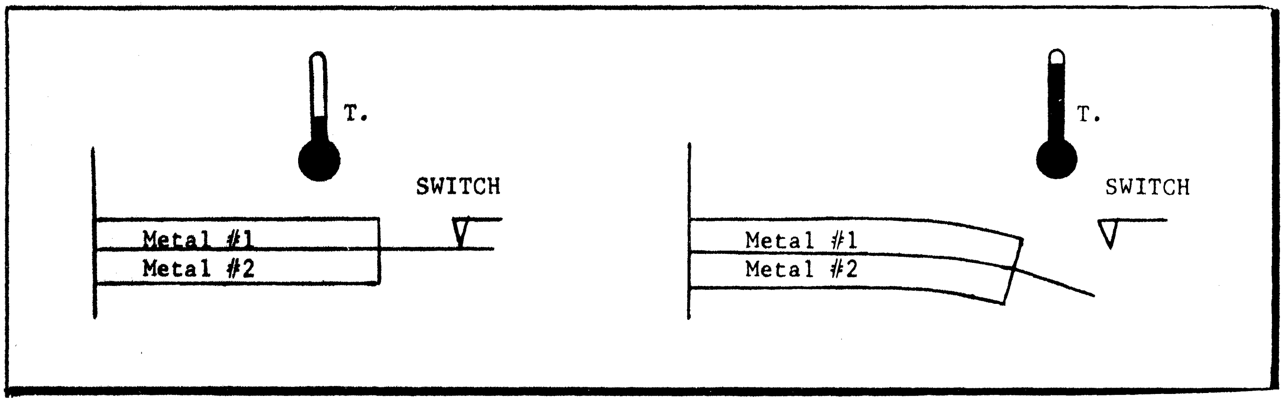


FIGURE 1

As you can see in the illustration, when the temperature is low the bonded unit does not warp. The switch connection is closed. This low temperature condition illustrates a method by which your furnace can be turned on. As the temperature rises, the bonded unit warps. Metal #1 expands faster than metal #2. As warping continues the switch connection is broken and for our illustration, the furnace is turned off.

Another example of analog conversion is the water meter in your home. When you turn the water in your home on, a small turbin-like wheel revolves in the stream of water entering your house. A small counter attached to the wheel begins to revolve and the number of revolutions are recorded on a counter. (Figure 2) This counter actually converts revolutions into the volume of water which flows through the meter.

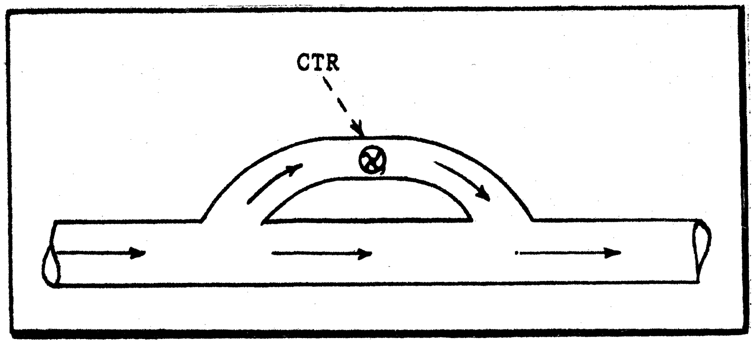


FIGURE 2

Similar examples can be repeated by many devices which you encounter daily. The electrical meter on the side of your house is built around a small induction motor. When you turn on a switch in the house, current begins to flow. This current induces a current in the meter and causes the meter to revolve. In a manner similar to our water meter, the revolutions of the electrical meter are converted into the amount of electricity used.

As you can see, physical phenomena such as temperature, revolution, flow, torque, and others are difficult for us to see and measure; therefore, a wide range of instruments have been developed to convert these analog representations into measurable form. These instruments appear in many forms; digital displays, charts, scale markings and dials. In Figure 3 we have used our bi-metal instrument with the addition of a needle indicator and graduated scale, and have illustrated the digitizing of an analog value.

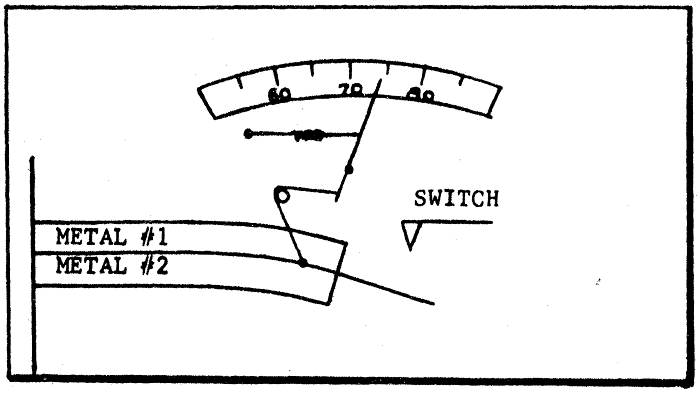


FIGURE 3

Using this simplified description of physical measurement, we can now proceed to develop a sample Industrial Control System.

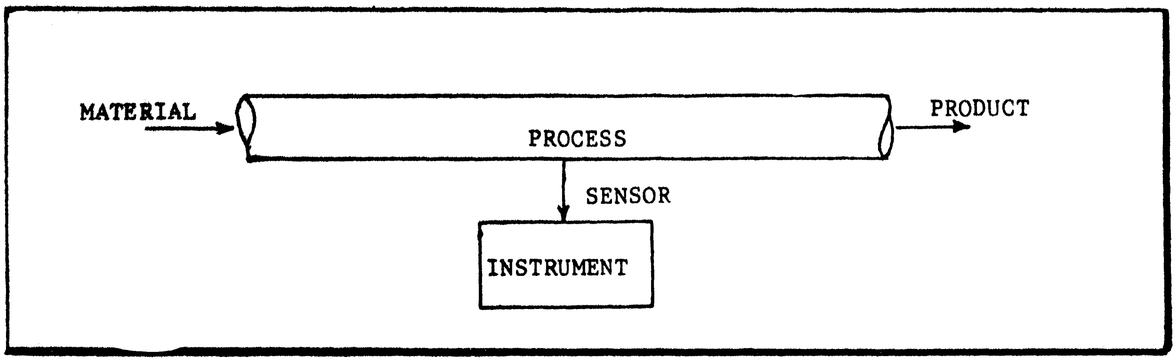


FIGURE 4

In Figure 4, we see only the measurement of a discrete property of a process. There are three components in our basic system; the PROCESS itself, a SENSOR, which measures the variable, and an INSTRUMENT to display this measurement. For example, the heat in our process can be measured by a simple mercury-filled thermometer. The mercury is our sensor. As the temperature in the process increases, the mercury expands and rises in the column of the thermometer. The instrument is the column in which the mercury rises and the display is the scale marking on the thermometer column from which we can read values for temperature.

We can expand our system by the introduction of a controller.

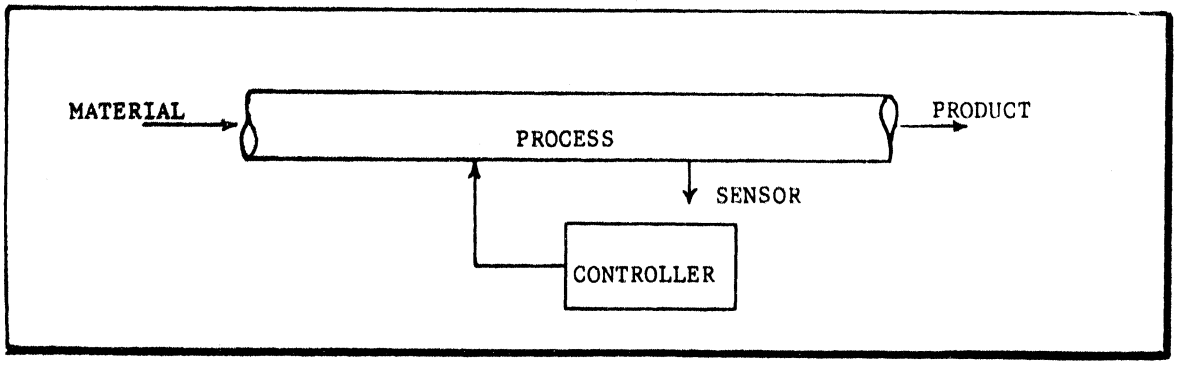


FIGURE 5

The controller has the capability of storing and comparing predetermined values with the value received from the sensor. To use the comparison results and develop our controller, we must add a control response element to our system. The capability of generating a response signal provides the means to use the result of this unit comparison and thereby creates a controller.

For example, using our bi-metal as a sensor, we can preset the gap in the switch contact. When the temperature in the process reaches a predetermined value called the set point, our bi-metal switch opens and the HEAT ON control signal sent to the process is discontinued (open switch) (Figure 6). As the process cools, our bi-metal returns to the closed switch position and again heat is added to the process.

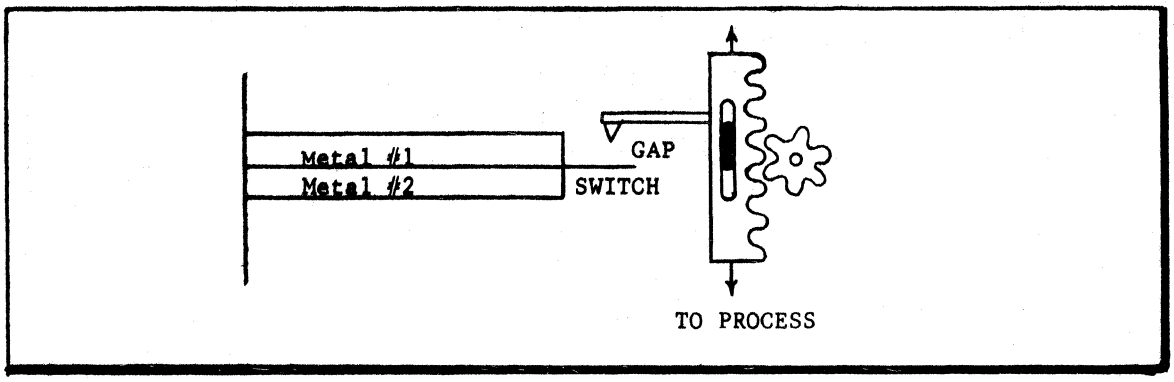


FIGURE 6

A further expansion of our control system should be the addition of some method of adjusting the set point value in the controller, rather than having the value permanently built into the controller. In the simplest form this could consist of a knob on the side of the controller.

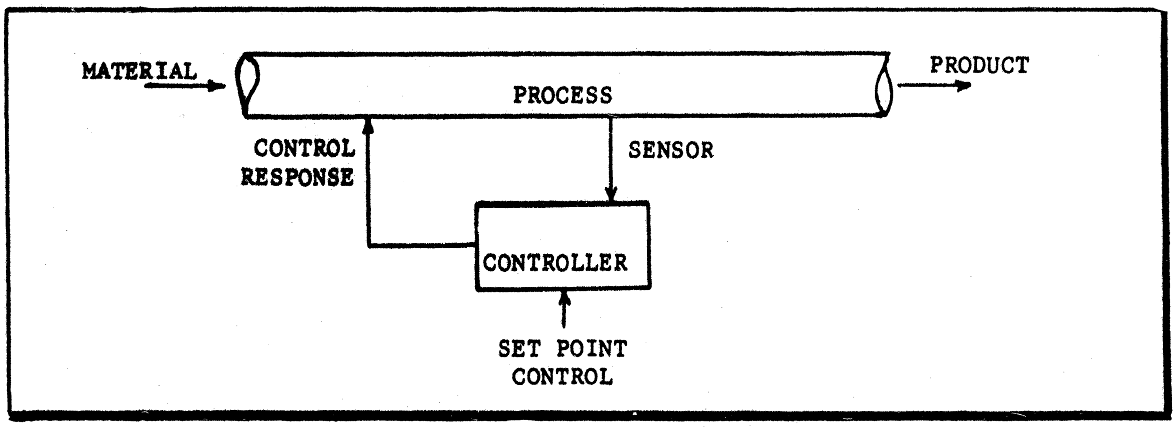


FIGURE 7

This knob would allow the changing of the set point. With this additional capability in the controller, an operator can adjust process variables and control the composition of the end product.

As the process becomes more complex, full process understanding and product improvement result from thorough study of all aspects of the process. In order to accomplish this process improvement effort, the process engineer needs continuous measurement and recording of the process variables. To continuously record process changes we can add components and build a data logging system.

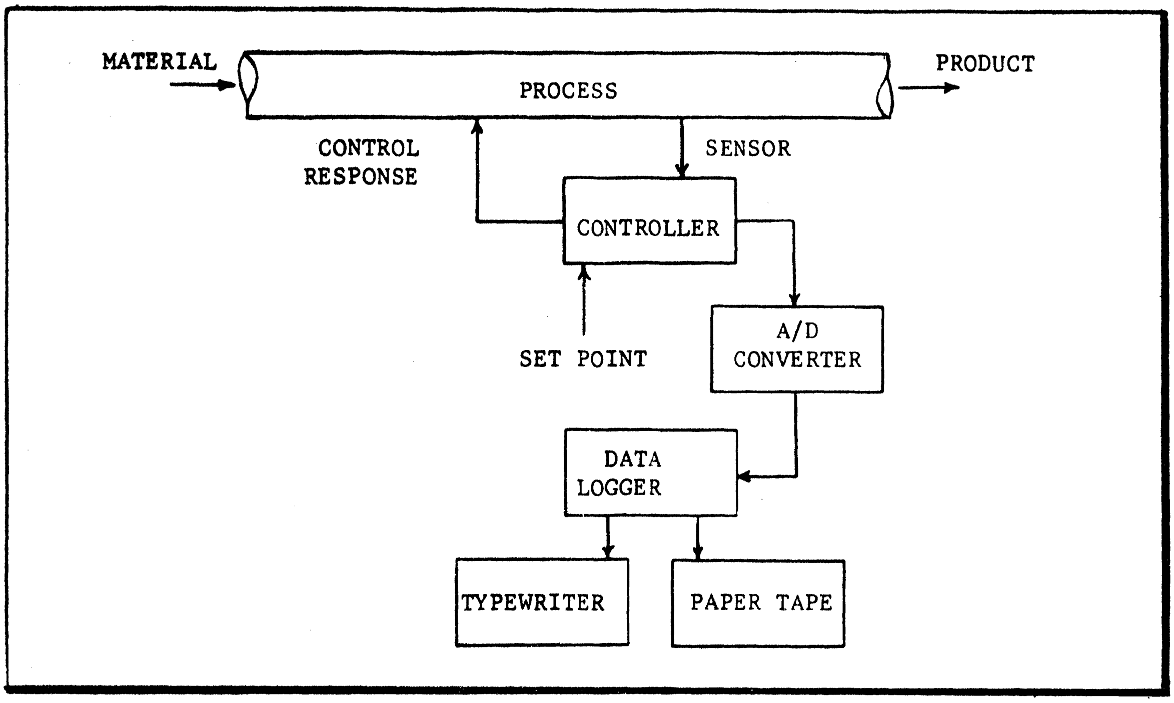


FIGURE 8

The heart of our data logging system is a small digital computer which converts, edits and prints out process variables determined by the sensor. However, in order to enter sensor data into the digital computer, we must make provision for the conversion of analog information into digital form. This conversion is accomplished by an analog to digital device.

A further requirement of our system is a component which will produce a permanent record of the values we are measuring. A typewriter or paper tape punch can well perform this function.

Now we have added (Figure 8) the necessary components which will permit our sensor to continuously transmit values into the data logger. At predetermined intervals controlled by the data logger, these values are recorded in permanent form by the typewriter or output device.

In order to achieve better control of the numerous variables in a process, we can further expand our system and accomplish the process control function automatically.

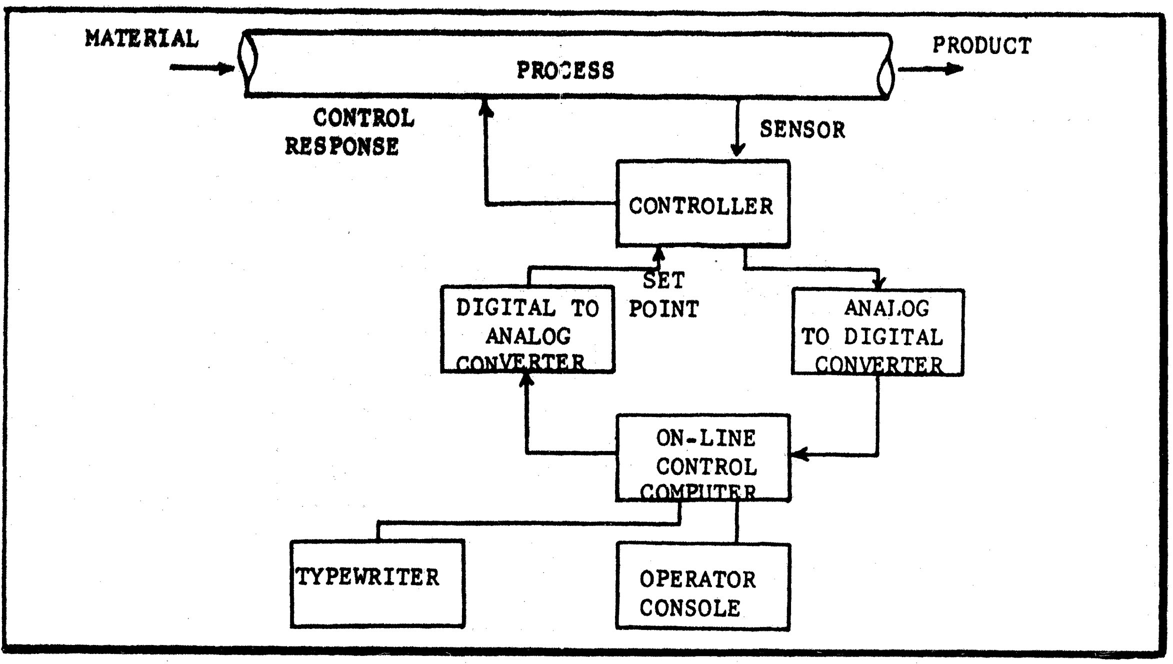


FIGURE 9

Using the data logging system, the operator necessarily interprets values as they are typed out, and manually adjusts controls to alter conditions in the process. By the addition of another component to our system, we can automatically alter control values.

In order to accomplish this function we require a digital to analog device; for example, a device which will provide a signal to move a valve or close a switch. In Figure 9 we show our closed loop system. By examination of the diagram you can see that a condition in the process sensed, converted, logged and is available to the control computer for decision making. The control computer then makes a determination; for example: Is the temperature too high, or too low? and transmits a signal to the controller which, in turn, generates a control response to the process.

In this system our process can proceed automatically. All factors which affect the process and product are monitored and adjusted as required.

As technologies improve, instrumentation techniques also improve. It is now possible to economically sense process conditions on devices which provide direct digital output signals.

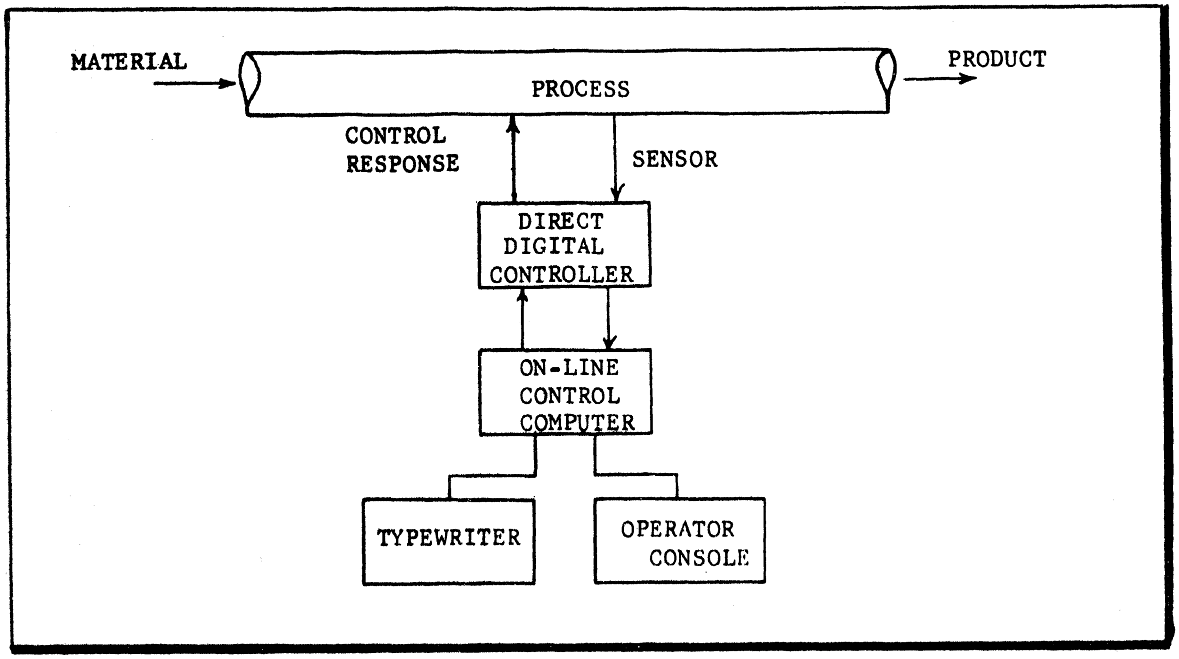


FIGURE 10

In Figure 10 we have shown the introduction of direct digital sensors into our process. As you observe by review of the diagram, we have eliminated the requirement for A/D conversion. The control sequence using direct digital sensors is functionally the same as described in our previous process.

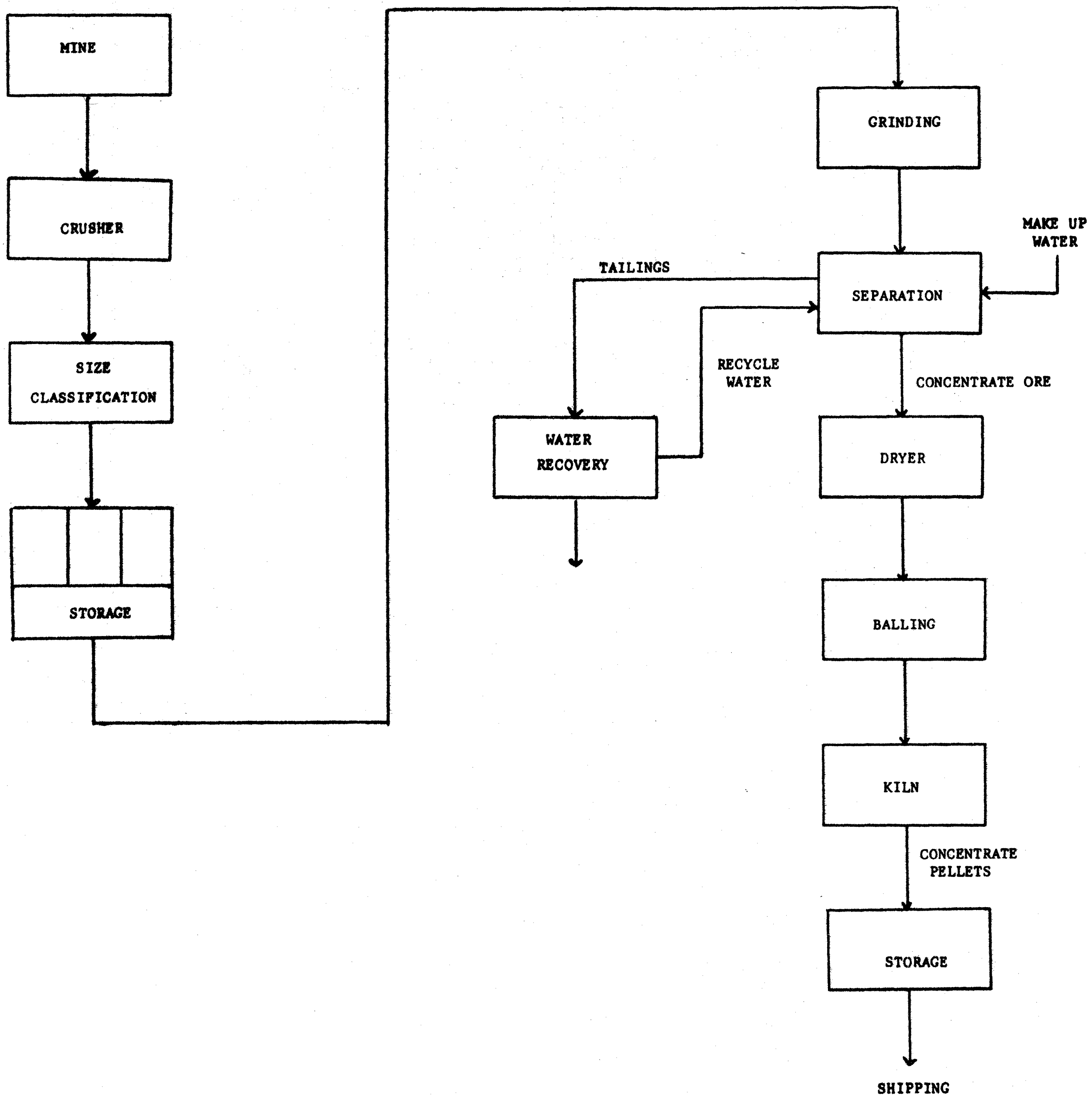
In many cases, direct digital control is more expensive than the previous analog methods; however, the repeatable accuracy, flexibility, and time sharing capability gained from the direct digital method can provide the economic justification required for its selection.

TYPICAL INDUSTRIAL COMPUTER SYSTEM APPLICATION

Although several segments of the metals industry have been among the leaders in the use of on-line computers, the mineral or ore processing function has not seen the application of on-line digital computers. Only in recent years has considerable interest developed in use of on-line digital control systems.

In the iron ore industry, blast furnace operating improvements have spurred developments in the preliminary ore processing. For example, blast furnace operations are much more efficient with pelletized raw materials than with conventional ores. Experiments in this area were spurred partially by the developments in the taconite industry in the upper midwest. Because of the demand by these blast furnace operations for pelletized ore, a number of ore beneficiation plants are being constructed in this country, in Canada, and overseas.

The processing operations in new plants approach the process complexity and large capital investment requirements of continuous chemical processes. It is not surprising, then, to see the mineral processing industry following the



TYPICAL FLOW DIAGRAM

IRON ORE BENEFICIATION & PELLETIZING FACILITY

FIGURE 11

lead of the chemical and petrochemical processors in the utilization of on-line digital computers to increase their process knowledge and to give better control of their processes.

Typical Ore Processing Plant

Figure 11 represents, in flow diagram form, a mineral processing facility typical of iron ore operations in a number of operating and planned-for plants. The prime purpose of a plant of this type is to increase the iron concentration of the ore to be shipped and to pelletize this ore before shipping to give best blast furnace operation. After the ore is blasted and removed from the ore body, the first operation in a beneficiation plant is to crush the ore into a more uniform size and provide a storage stock pile with classification of sizes and grades.

The next stage is a grinding operation to provide a fine powder. The separation of iron from rock may be done in many steps using the principles of (1) magnetic separation in which the magnetic iron particles are separated from the non-magnetic silica, (2) centrifugal forces may be used in which the separation is achieved by the difference in density of the iron rich ore and the base material, and (3) electrostatic potentials may be utilized to perform a separation based upon the different electrical characteristics of the desired ore and the base.

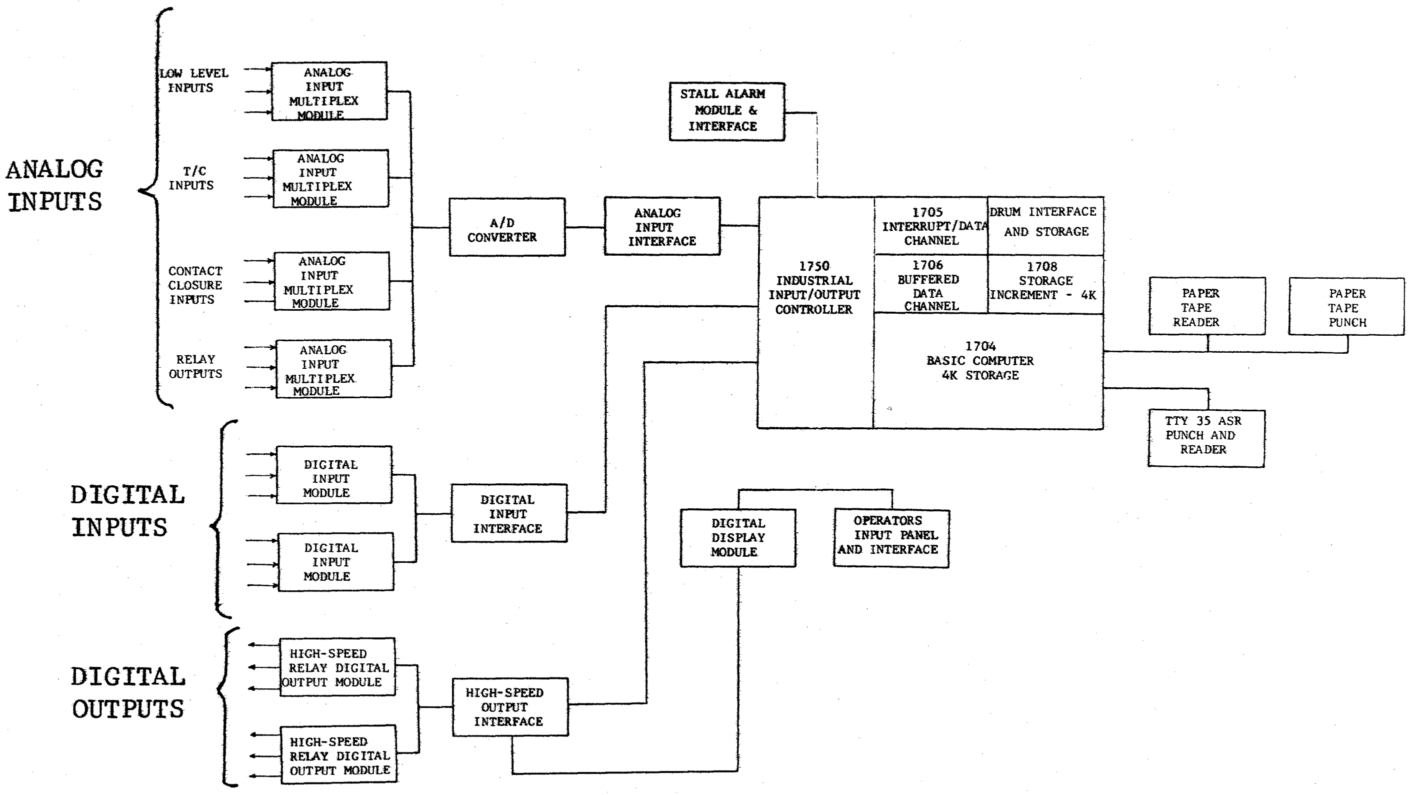
Enriched ore is sent on to a drying stage and then to a pellet-making operation in which the iron particles are mixed with clay materials and formed into pellets. These pellets are then fired at very high temperatures in a fluidized bed furnace or in a traveling grate furnace to sinter the pellets. The pellets are now ready to be shipped to the blast furnace.

All along the line, a considerable amount of waste material is generated and must be disposed of properly and according to modern land and water conservation principles. Extensive amounts of water are required by these plants, and a considerable part of the plant is devoted to water recovery and waste removal.

A number of variables are of interest to operating personnel of these plants and include many of the following: ore flows, water flows, material inventories, chemical composition of various flows, power per ton of material processed, metal recovery from the raw ore (process yield), production rates (comparing shifts or daily, weekly, and monthly averages), operating hours on heavy equipment, equipment availability and many temperatures, fuel flows, product flows, etc. These variables represent the basic information inputs to an operational information system utilizing a digital computer.

Typical Computer System Configuration

Figure 12 is a configuration of a typical Control Data Operational Information System and shows some of the functional modules required in this equipment and approximate numbers of signals commensurate with a mineral processing plant. The system receives on/off or pulse inputs from alarm relays, belt scales which weigh various solid material flows, and other transducers generating digital on/off information. A typical system might include 500 to 1000 such inputs.


ANALOG INPUTS
LOW LEVEL INPUTS
ANALOG INPUT MULTIPLEX MODULE
T/C INPUTS
ANALOG INPUT MULTIPLEX MODULE
CONTACT CLOSURE INPUTS
ANALOG INPUT MULTIPLEX MODULE
RELAY OUTPUTS
ANALOG INPUT MULTIPLEX MODULE
A/D CONVERTER
ANALOG INPUT INTERFACE
STALL ALARM MODULE & INTERFACE
1750 INDUSTRIAL INPUT/OUTPUT CONTROLLER
1705 INTERRUPT/DATA CHANNEL
DRUM INTERFACE AND STORAGE
1706 BUFFERED DATA CHANNEL
1708 STORAGE INCREMENT - 4K
1704 BASIC COMPUTER 4K STORAGE
PAPER TAPE READER
PAPER TAPE PUNCH
TTY 35 ASR PUNCH AND READER
DIGITAL INPUTS
DIGITAL INPUT MODULE
DIGITAL INPUT MODULE
DIGITAL INPUT INTERFACE
DIGITAL DISPLAY MODULE
OPERATORS INPUT PANEL AND INTERFACE
DIGITAL OUTPUTS
HIGH-SPEED RELAY DIGITAL OUTPUT MODULE
HIGH-SPEED RELAY DIGITAL OUTPUT MODULE
HIGH-SPEED OUTPUT INTERFACE

TYPICAL CONTROL DATA 1700

OPERATIONAL INFORMATION SYSTEM

FIGURE 12

Analog measurements of temperature, flow rates, and similar variables are essential input information and are entered through analog input multiplexers. Usually there are both high level and low level signals coming in to such a system. Modern electronic instruments generate primarily high level signals and represent the bulk of the inputs to such a system, however, thermocouples and similar devices generate millivolt readings and are often brought directly into the system. Again, these analog signals are multiplexed, amplified, and converted to digital signals in an analog to digital converter. A typical system might contain hundreds of analog inputs.

The digital input information and the digitized analog information is sent into the computer through some form of computer input/output interface connection. The input/output interface is under control of the digital computer and contains the circuitry required to decode the information from the computer as to what signal it is requesting and to transmit data to or from the computer. Digital computers utilized in these systems generally use a basic core memory for high speed arithmetic and data manipulation. This memory is quite often supplemented by a drum or disc memory offering greater bulk storage at lower cost. This type storage contains information to be logged periodically, averaged, etc. A number of logging typewriters are desired for such things as shift logs, daily logs, alarm logs, and other operational information logs required by operating personnel.

There is also a necessity for an operations console which allows operating personnel to insert information and receive information from the computer on an as-desired, point by point basis rather than by complete logging. Control of the process through the computer either initiated by the operator or by a computer program is accomplished by routing signals back to the process either as discrete output signals such as relay closures which can raise or lower motors, etc., or as analog information which comes from digital to analog converters.

Often systems of this type represent only sub-loops in an overall management control system for a large intricate plant, in which case it is necessary to have teletype or telephone data communication links to other computers at this location or remote sites.

Features

High speed alarm scan and log.

Shutdown sequence determination.

Pulse counting.

Logging operator actions.

Analog scan with high-low limit comparisons.

Real time clock.

Input variable averages (hourly, daily - - - yearly)

On-line connection to analytical laboratory.

Composition calculations weighted by flow rates.

Operator entry under program control.

Periodic and demand logs.

Typewriter or paper tape output.

Control system failure monitor.

Performance calculations written in FORTRAN.

Control outputs (raise-lower contacts or D/A converters).

Equipment availability calculated and operating times accumulated.

Trend recording.

Post-failure review (process history).

Operational performance calculations and management reports.

1700 CHARACTERISTICS

The Control Data 1700 computer is a small, stored-program, parallel mode digital computer designed to meet modern demands for fast, low-cost computation and control. Unique hardware features, coupled with a broad range of programming packages, make the 1700 a powerful, versatile tool for industrial control, data acquisition and communications uses.

The 1700 is characterized by:

1. Advanced circuits which give the 1700 a storage cycle time and instruction execution times well below those of many large computer systems.

2. All-silicon components and ruggedized construction enabling the 1700 to operate under varying and difficult environments with no special adjustments necessary.

3. Its ability to communicate not only with standard peripherals (tape units, printers, card readers, and the like), but also with many types of specialized industrial control and communications devices. A direct access connection to storage facilitates high-speed data transfers.

4. The ease with which the system may be expanded as the user's needs dictate. The enormous computing power of the Control Data 3000 and 6000 series computers can be combined with the 1700 by means of a special adapter connecting the main frames.

5. Its software providing the capability to handle a real-time and a conventional program simultaneously.

1700 SPECIFICATIONS

Hardware

18-bit storage word (16 data bits, Program Protect bit, parity bit)
16-bit instruction word
4096-word basic storage, expandable to 32,768 words
1.1 microsecond storage cycle time
Program Protect System
Parity checking
2 indexable registers
Multi-level indirect addressing

16-level priority interrupt system, internal and external interrupts
Hardware-buffered and interrupt-buffered Input/Output
One's complement, signed arithmetic
Fixed-point Add, Subtract, Multiply, Divide
Control Data 6600-type silicon circuits
Components cooled by room air
Console switches and indicators
Standard non-buffered I/0 bit rate: 1.4 million bits per second

Software

The capabilities of the 1700 computer system are exploited to the fullest by an advanced library of software routines. All software can be operated on-line on a time-shared basis. Standard software catagories are as follows:

Real-Time Operating System
Modular Industrial Process Control Packages
Symbolic Assembler
Macro Assembler
Fortran Compiler
Time-sharing monitor
Arithmetic Package
Utility Routines

1700 APPLICATIONS

Industrial Process Control

The logical design and physical specifications of the 1700 computer are pointed toward making it adaptable to the vast number of uses and environmental conditions encountered in industry.

It is ideally suited to serve such diverse areas of industry as conventional and nuclear power stations, steel mills and other metals, chemical productions, and oil and gas pipe lines. Among the functions which a 1700 control system is capable of performing are:

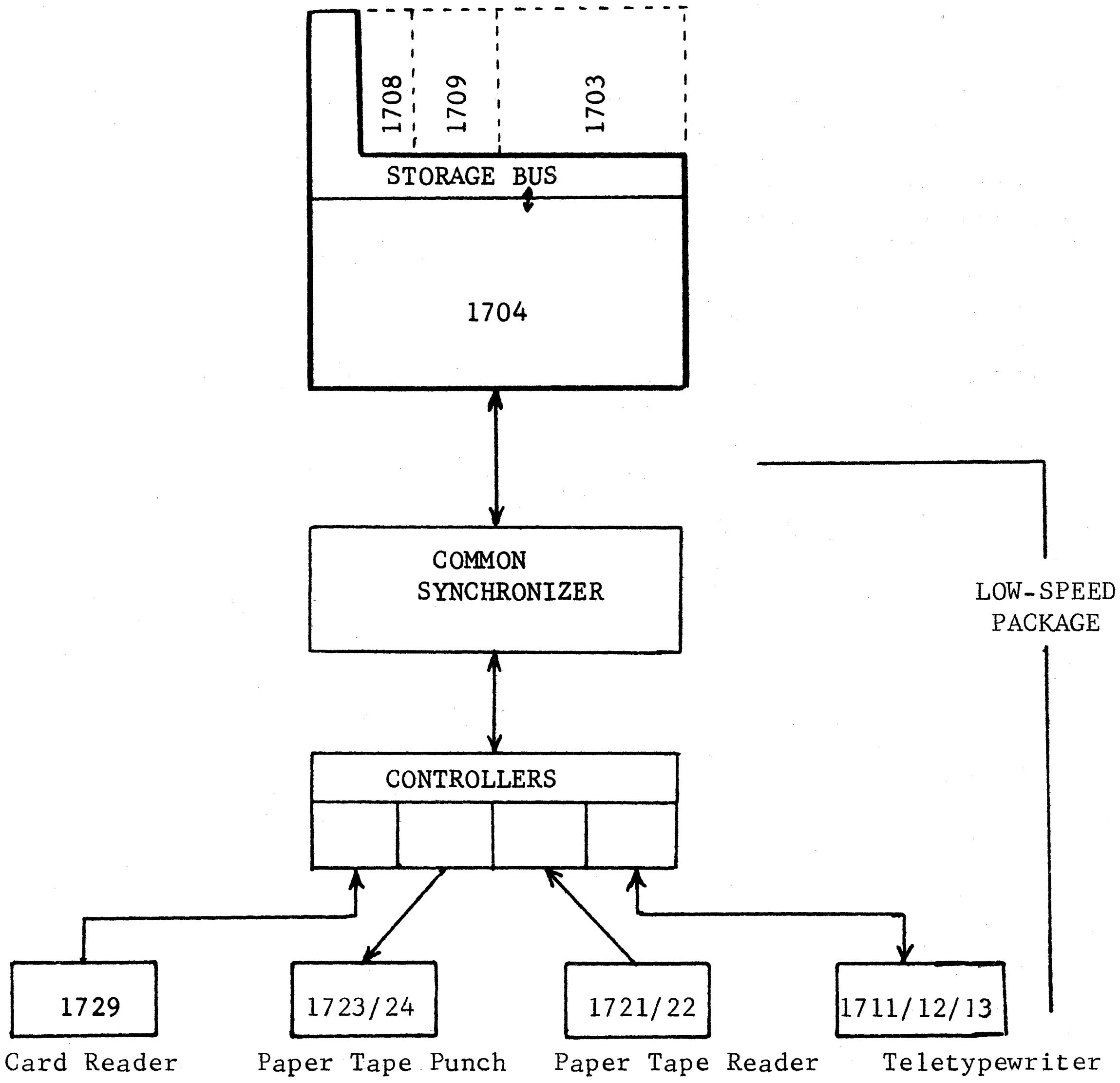
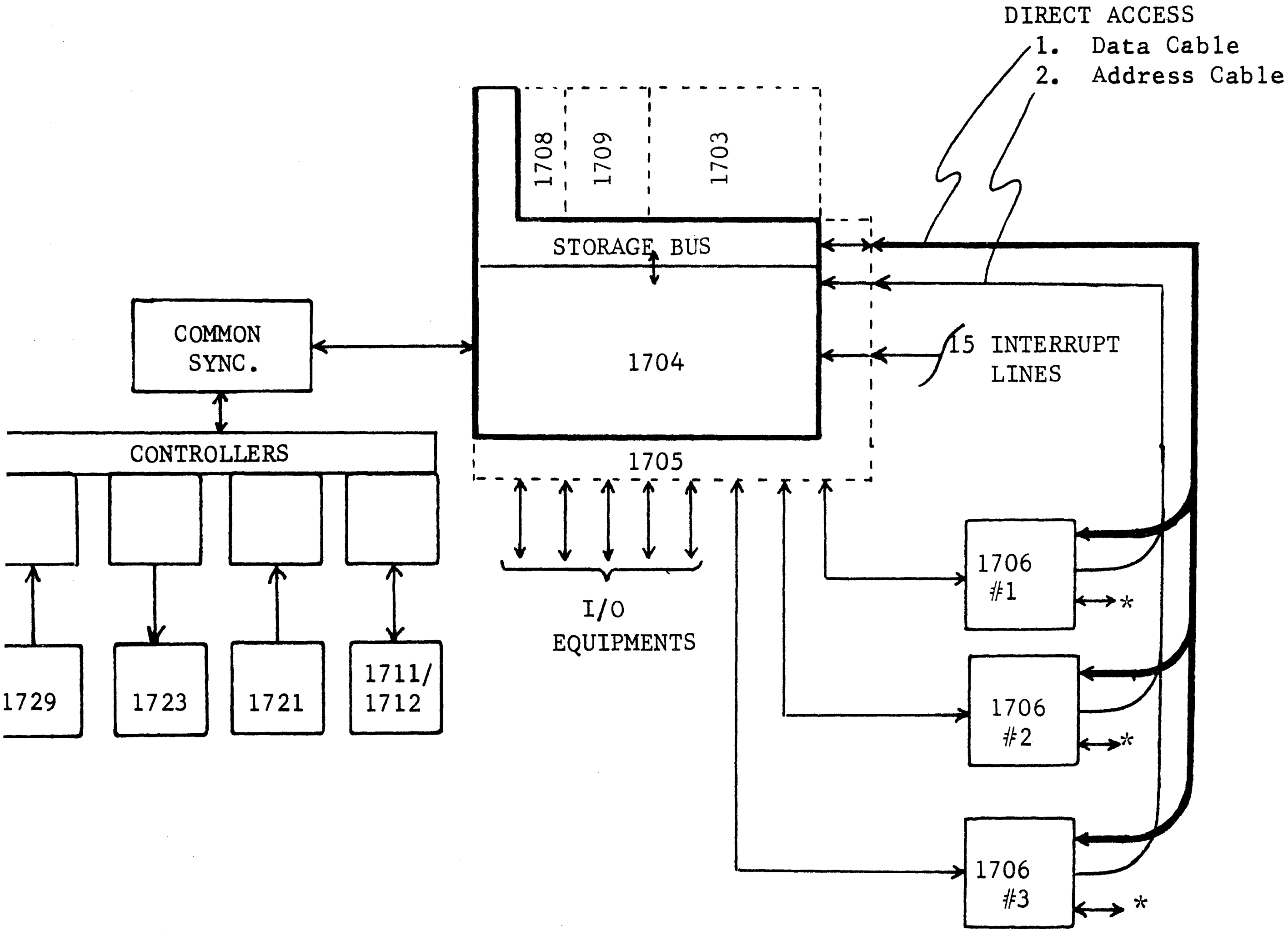


FIGURE 13

1700 BASIC COMPUTER SYSTEM



* Eight I/O Equipments Maximum

FIGURE 14

1700 EXPANDED COMPUTER SYSTEM

1. Logging
2. Data Display
3. Alarm Monitoring
4. Direct Digital Control
5. Performance Calculations
6. Remote Supervisory Control
7. Event-Oriented Control
8. Report Preparation

Data Acquisition and Conversion

The extremely high speed of the 1700 Computer has special relevance for the field of research and development. Through the use of analog-to-digital and digital-to-analog conversion equipment, combined with the high-speed parallel-processing capabilities of the computer, an opportunity for laboratory analysis hitherto unfeasible with digital computers exists.

Areas in which the speed and versatility of the 1700 system may be applied include medical research, wind-tunnel and aerospace research, auto-correlation, radar and sonar studies, analysis of vibration and resonance in physical structures, and atomic research.

Communications and Data Collection

The 1700 Computer, by virtue of its speed and input/output flexibility, meets the requirements of message-switching and collection systems, where large volumes of data must be transmitted over great distances in a minimum of time.

The 1700 may function as a self-contained communications center or as a high-speed buffer, receiving data from remote terminal stations and routing it to a large central computer for further processing.

MODEL DESCRIPTION

One of the outstanding features of the 1700 is the ease with which storage and I/O capabilities may be expanded. Following are brief descriptions of the standard and optional units.

1704 A 1700-Class computer with the following features: arithmetic, including single-precision Multiply and Divide; basic 4096-word storage; non-buffered output to the Common Synchronizer; 8-bit teletype communication. It includes one internal and one low-speed interrupt.

1705 Interrupt/Data Channel. This option increases the I/O capability of the 1704 Computer. The 1705 enables reading from and writing into storage via the A and Q registers and the direct access bus. It adds 15 external interrupt lines increasing the capability by 14 Interrupt levels to a total of 16. It implements buffered and non-buffered I/O transfers.

1706 Buffered Data Channel. This option enables buffered I/O operations. It connects to the 1705 Interrupt/Data Channel and to the direct access bus to storage. For non-buffered I/O, only the 1705 need be added to the basic computer; for buffered I/O, both the 1705 and 1706 are necessary. A maximum of 3 1706's may be connected to a computer.

1708 A 4096-word storage increment which may be added to a 1704 Computer, giving a total storage capacity of 8,192 words.

1709 An 8,192-word storage increment which may be added to a 1704 Computer to which a 1708 storage option has been added, giving a total storage capacity of 16,384 words.

1703 A 16,384-word storage increment which may be added to a 1704 Computer to which a 1708 and a 1709 storage increment has been added, giving a total storage capacity of 32,768 words for a 1704 Computer.

Storage Size*	Models Required
4k	1704
8k	1704, 1708
16k	1704, 1708, 1709
32k	1704, 1708, 1709, 1703

*Storage capacities of 12k and 24k are also available as special options.

1716 Coupling Data Channel. This option permits communication between two 1700 Computers and also enables the two computers to have access to the same peripheral devices via the 1705 Interrupt/Data Channel. When the 1716 is used, only two 1706's may be connected in a maximum system.

Common Synchronizer A Card/Paper Tape Data Channel for direct interface into the A and Q registers of the 1704 without the need for a 1705 option. The following devices may be connected to this synchronizer: the 1721 Paper Tape Reader, the 1723 Paper Tape Punch, the 1729 Card Reader, and the 1711 (or 1712, 1713) Teletypewriter.

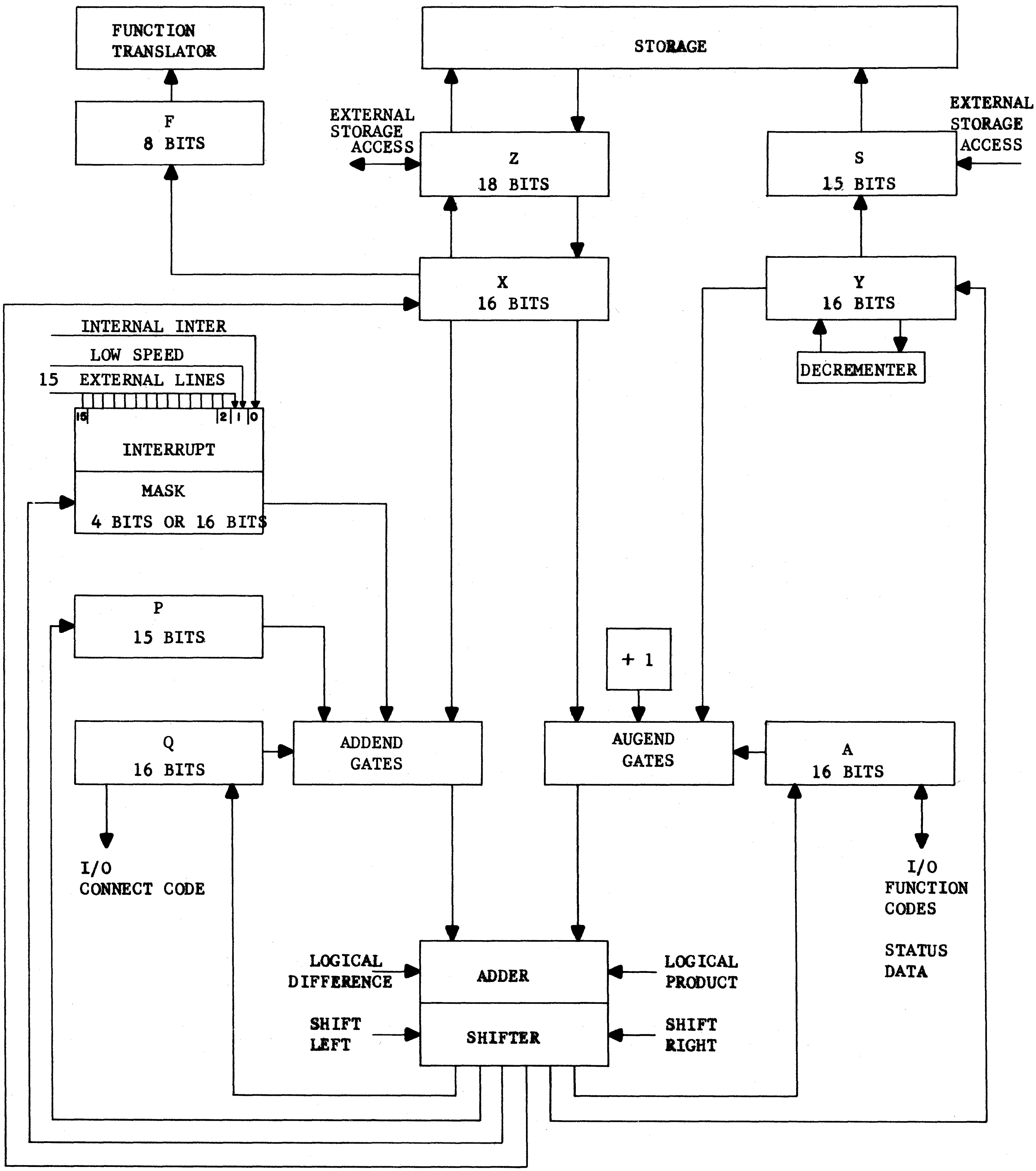


FIGURE 16 Block Diagram

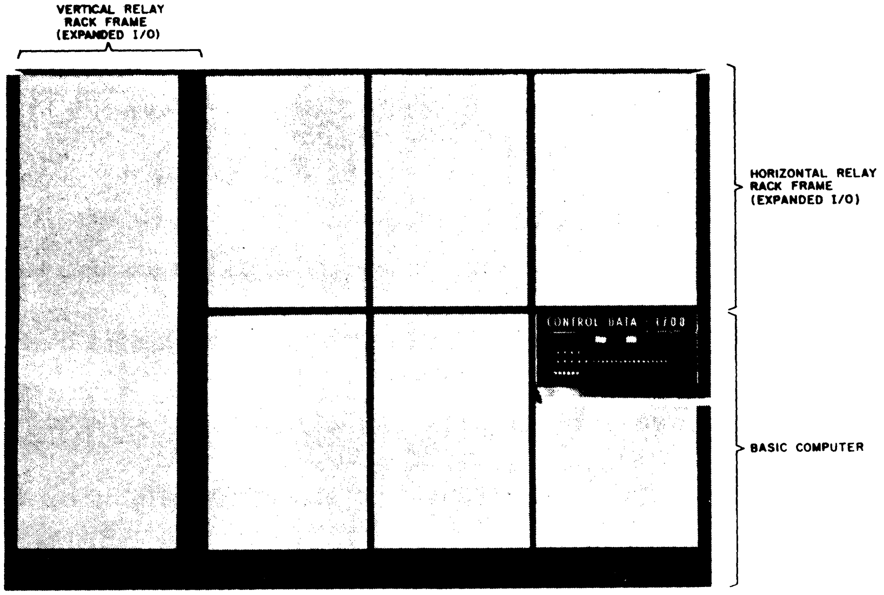

VERTICAL RELAY RACK FRAME (EXPANDED I/O)
HORIZONTAL RELAY RACK FRAME (EXPANDED I/O)
BASIC COMPUTER
CONTROL DATA 1700

FIGURE 15 1700 Computer System with I/O Modules

1704 Registers See Figure 16

A Register

The principal arithmetic register. It contains 16 bits, with the 16th bit being the sign bit. The results of most arithmetic and logical operations are placed in A. It also serves as a data register for I/O transfers.

Q Register

A 16-bit auxiliary arithmetic register. It contains address information during I/O instructions and is also used as an index register.

P Register

The 15-bit program address register. It holds the address of each program step.

X Register

The 16-bit exchange register. It holds data going to or from storage and one of the parameters in most arithmetic operations.

Y Register

A 16-bit register. Storage addresses are formed and held here for transfer during a storage reference. It is also used as a counter during Multiply, Divide, and Shift instructions.

F Register

The 8-bit function register. This register holds the instruction identification and/or addressing mode during instruction execution.

Z Register

The 18-bit storage data register. It transfers data to or from the computer, the external access, and storage. Each time a word is read from storage it is transferred to Z and parity is checked. Word parity is generated when data is in Z prior to being written into storage.

S Register

The 15-bit storage address register. This register holds the address of the word being read from or written into storage. The upper 3 bits designate one of eight 4K storage modules; the remaining bits specify an address within the selected storage module.

Mask Register

This register is the enable for the interrupt levels. It is either 4 or 16 bits. To select an interrupt on a given level, the corresponding Mask bit must be set.

PROGRAMMING AIDS

In the appendix are various tables which are intended to save time in reading or writing 1700 programs. Tables I and II provide numerical and alphabetical listings of 1700 instructions while Table III gives all possible methods of forming the effective address and lists the location of the next instruction for each addressing mode. Parentheses mean "the contents of" whatever is in the parentheses. For example: (Q) means the contents of the Q Register, (P+1) means the contents of location P + 1 (since P + 1 does not indicate a register) and (+ △) means the contents of location + △ (again, since + △ does not indicate a register). A good example of the use of parentheses is the formation of the effective address (E.A.) if △ = 0 and the addressing code is "F". In this instance, the E.A. is formed by (P+1 + (P+1) + (Q) + (00FF) which says, to form the effective address:

1. Read the contents of the location specified by the quantity P+1 added to the contents of location P+1.

2. To the result in #1, above, add the contents of the Q Register.

3. To the result in #2, above, add the contents of location 00FF.

If two hexadecimal numbers are to be added, use Table IV and find one digit along the left and the other digit along the top of the table. The hexadecimal sum is the quantity contained in the area where the selected column and row intersect. If the numbers being added contain several digits each, consider one pair of digits at a time keeping track of the carries generated. Table IV can also be used for subtraction by locating the subtrahend digit along the top of the table then going down the column until the minuend digit is found in the body of the table. The difference is the number of the row (left-hand column) where the minuend is found. Whenever a digit is being subtracted from one of smaller value, the minuend will be a 2-digit number (due to the borrow).

Table V can be used to perform hexadecimal multiplication or division. Use of the table is similar to Table IV. Be certain that additions (for multiplication) and subtractions (for division) are also performed in hexadecimal.

HEXADECIMAL NUMBER SYSTEMS

The 1700 uses the hexadecimal (base 16) numbering system to represent it's data and instructions. A review of the decimal, binary, and hexadecimal systems alon with useage of hexadecimal arithmetic will be useful at this time. As you proceed through the chapters on programming an increasing amount of hexadecimal arithmetic will be encountered. Instructions are coded in hexadecimal, numbers are displayed in binary but are converted to hexadecimal by the computer operato for the sake of making comparisons with anticipated values after a program is ru Storage addresses are coded in hexadecimal and are added to or subtracted from in hexadecimal. A review of the number systems used in the 1700, their conversi to other bases, and hexadecimal arithmetic will follow in the succeeding paragra

A look at the three number systems which will be encountered will be the first subject. It will be assumed that you have a knowledge of binary at this time. Counting in the three systems goes something like this:

Decimal	Binary	Hexadecimal
0	0000 0000	0
1	0000 0001	1
2	0000 0010	2
3	0000 0011	3
4	0000 0100	4
5	0000 0101	5
6	0000 0110	6
7	0000 0111	7
8	0000 1000	8
9	0000 1001	9
10	0000 1010	A
11	0000 1011	B
12	0000 1100	C
13	0000 1101	D
14	0000 1110	E
15	0000 1111	F
16	0001 0000	10
17	0001 0001	11
18	0001 0010	12
19	0001 0011	13
20	0001 0100	14

Notice that after the hexadecimal 9 the letter A is used to represent 10. A through F are used to represent the last six numerals of the hexadecimal system. Also, hexadecimal 10 follows the number F in the numerical sequence. It might be well to note at this time that the number 7FFF would be represented in the 1700 as a binary number *111 1111 1111 1111. The * position is the sign. If a "0" occupies that position the number is positive, but if it is a "1", then, the number is negative.

HEXADECIMAL TO DECIMAL CONVERSION

The methods of conversion from decimal to hexadecimal and hexadecimal to decimal will be useful before starting into the instructions and programming. The conversion from hexadecimal to decimal can be done either of two ways. The first example will use the polynomial method of conversion. The number 7A46 will be used as an example.

7A46

$\rightarrow 16^0$

$\rightarrow 16^1$

$\rightarrow 16^2$

$\rightarrow 16^3$

each number has positional value as shown.

Once the positional values have been written down, expand them by raising the base (16) to the amount indicated by the exponent. After this is done, multiply the number times its positional value.

7A46

$\rightarrow 16^0$	=	1	x	6	=	6
$\rightarrow 16^1$	=	16	x	4	=	64
$\rightarrow 16^2$	=	256	x	A	=	2560
$\rightarrow 16^3$	=	4096	x	7	=	28672

$7A46_{16} = 31{,}302_{10}$

Remember that the A has to be converted to 10 before it can be multiplied times the positional value of 256. If the most significant digit is larger than 7 the number is negative. To convert it, it must be complemented and then be converted.

$FF67_{16} = -0089_{16}$

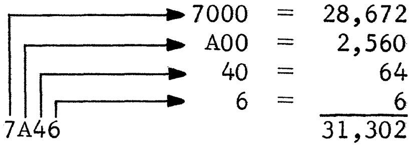
Complement the number by subtracting FF67 from FFFF and affix the minus sign to the complemented number. The example, -0098, can now be converted to decimal.

-0098

$\rightarrow 16^0$	=	1	x	8	=	8
$\rightarrow 16^1$	=	16	x	9	=	144
						152

$-0098_{16} = -152_{10}$

Another method of converting from hexadecimal to decimal is with the aid of the conversion chart in the appendix. Using the same number used in the first example, a conversion will be made using the chart.



Start by looking up the most significant digit, 7000, in the fourth hexadecimal column. Continue looking up t he remaining digits in sequence in columns 3, 2, and 1. Add up the conversions for each digit.

Convert the following hexadecimal numbers to their decimal equivalent.

Hex. Decimal

1. 53,FF6 ____________________

2. 10,04E ____________________

3. 579,F65 ____________________

4. 567,999 ____________________

5. 48A,000 ____________________

6. FACE ____________________

7. CAFE ____________________

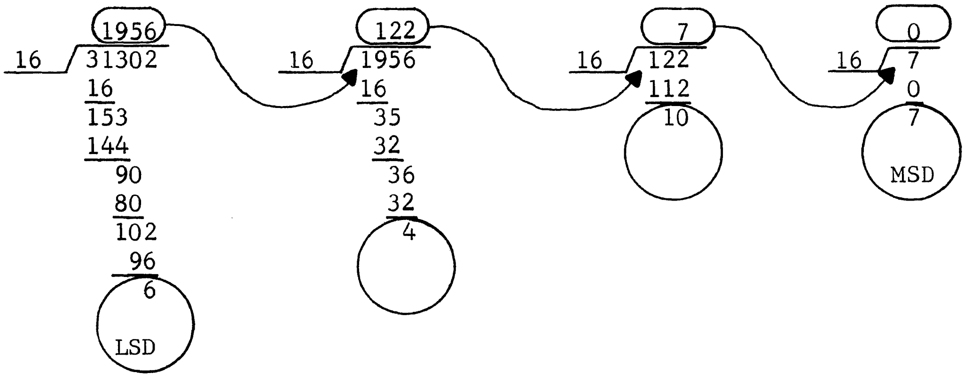
8. DDE,904 ____________________

9. FA,875 ____________________

10. F6,45D,6FA ____________________

DECIMAL TO HEXADECIMAL CONVERSION:

Conversion from decimal to hexadecimal is also necessary. Again, there are two methods; one by division, the other by the use of the conversion chart. In the event that a conversion chart is not handy it may be well to illustrate the division method.



$7A46_{16}$

The conversion is made by dividing 16 into the decimal number to be converted. The remainder is the least significant digit of the hexadecimal number. The dividend is then divided by 16 to produce the next digit. Continue doing this until the conversion has been completed. This method may take more time, but it doesn't require any tables or charts to make the conversion. However, if a conversion chart is handy, it will save considerable time. The number 31,302 will again be use for illustration.

```
31,302
28,672 . . . . . . . . . 7000
 2,630
 2,560 . . . . . . . . . .A00
    70
    64 . . . . . . . . . . 40
     6
     6 . . . . . . . . . . .6
                         7A46
```

Using the conversion chart on page A-26 of the appendix, it is found the 28,672 is the closest decimal number to 31,302 without exceeding it. Subtract the 28,672. from 31,302 and write down the hexadecimal 7000 for future use. Next, look up the decimal number that is equal to or less than the difference between 31,302 and 28,672 (2,630). It can be found in the third column of the decimal numbers. Again, a subtraction is made between 2,630 and 2,560 resulting in a difference of 70. The A00, hexadecimal equivalent of 2,560, is added to the hexadecimal 7000 to form an additional portion of the conversion. Continue with the conversion until it is completed. When making conversions, use whichever method you wish, keeping in mind that the final result is the important thing.

Convert the following decimal numbers to hexadecimal.

Decimal   Hexadecimal

1. 24,088 ________________

2. 76,744 ________________

3. 524,288 ________________

4. 1,234,567 ________________

5. 7,633,994 ________________

6. 56,744,485 ________________

7. 1,357,998,765 ________________

8. 1,879,048,191 ________________

9. 1,000,000,000 ________________

10. 3,333,333,333 ________________

ADDITION

The next subject will be hexadecimal arithmetic. Finger counting is permissible, but the chart for hexadecimal arithmetic on page A-24 will be more useful. As an example, $C405_{16}$ and 8045_{16} will be added together.

```
                     8,045
                     C,405
                    ------
                    14,44A
end-around-carry         1
                    ------
                     4,44B
```

Use the chart on page A-24 as an aid. First find the number 5 on the left hand margin. Move across from that point until you are under the 5 at the top of the chart. Your finger should now be on the number A. The 4 and 0 for the next two additions should be no problem. When finding the sum of 8 and 6 use the same procedure that was used to find the sum of 5 and 5. Locate the 8 on the left margin. Move to the right until your finger is on the number in the column headed by the number C. As indicated by the chart the sum of C and 8 are 14. The 1 is carried around and added to the least significant digit. You cannot exceed four hexadecimal numbers as the result of an addition. The adder will accomodate only the four numbers. If a thought came to you that this was overflow you are wrong. Overflow can occur only when two like signed numbers are added together and the sum results in a change of sign. Remember that numbers larger than 7 would indicate that they are negative. An illustration of overflow can be illustrated by the following examples.

$$\begin{array}{rl} \text{7FFF} & \text{positive} \\ \underline{\text{0001}} & \underline{\text{positive}} \\ \text{8000} & \text{negative} \end{array}$$

$$\begin{array}{rl} \text{8000} & \text{negative} \\ \underline{\text{8000}} & \underline{\text{negative}} \\ \text{1 0000} & \\ \underline{\text{1}} & \\ \text{0001} & \text{positive} \end{array}$$

Solve the following hexadecimal addition problems.

$$\begin{array}{llllllllll} 1. & \text{5C02} & 2. & \text{AF02} & 3. & \text{2377} & 4. & \text{8100} & 5. & \text{ABCD} \\ & \underline{\text{0227}} & & \underline{\text{1245}} & & \underline{\text{15FF}} & & \underline{\text{9011}} & & \underline{\text{1234}} \end{array}$$

$$\begin{array}{llllllllll} 6. & \text{0006} & 7. & \text{0105} & 8. & \text{CAFE} & 9. & \text{FACE} & 10. & \text{F07A} \\ & \underline{\text{FFFE}} & & \underline{\text{CCCC}} & & \underline{\text{BCDE}} & & \underline{\text{4333}} & & \underline{\text{C5D3}} \end{array}$$

SUBTRACTION:

Subtraction is a reverse process of addition. Subtracting 3243_{16} from $7AB6_{16}$ can illustrate the process of subtraction.

$$\begin{array}{r} \text{7AB6} \\ \underline{\text{3243}} \\ \text{4873} \end{array}$$

Locate the number 3 on the left side of the addition chart. Move to the right until you come to the number 6. The number at the top of that column will be the difference between 3 and 6. Next locate the number 4 in the left hand column. Move to the right until you come to the B. The number at the top of that column is the difference between 4 and B. Continue until the remainder of the problem is completed.

Solve the following hexadecimal subtraction problems.

$$\begin{array}{llllllllll} 1. & \text{3210} & 2. & \text{1600} & 3. & \text{F07A} & 4. & \text{3789} & 5. & \text{FFFF} \\ & \underline{\text{1022}} & & \underline{\text{00FF}} & & \underline{\text{609F}} & & \underline{\text{308D}} & & \underline{\text{7ACE}} \end{array}$$

$$\begin{array}{llllllllll} 6. & \text{CAFE} & 7. & \text{4567} & 8. & \text{FACE} & 9. & \text{2D6E} & 10. & \text{5BD6} \\ & \underline{\text{ABCE}} & & \underline{\text{3F4D}} & & \underline{\text{9D6B}} & & \underline{\text{1A7F}} & & \underline{\text{3A21}} \end{array}$$

MULTIPLY

Occasionally it may be necessary to check a multiply problem in the 1700. If you can work out the problem on paper, you can check the product which the computer is to come up with. A couple of points which must be kept in mind when doing a multiply; overflow can never occur, and negative numbers are complemented before the multiply operation takes place. A four digit hexadecimal number is multiplied times a four digit number. The product will be an eight digit hexadecimal number. Negative numbers are complemented before the multiply operation because multiplication is the process of finding the product of absolute values (absolute values are expressed in a computer only by positive numbers). After the multiply operation is completed the product is complemented if only one of the operands was negative.

The multiply chart on page A-25 will be needed to do multiplication. Use the same procedure used when multiplying decimal numbers. Remember to add the carries to the next partial product as you proceed. Suppose the number 45 is multiplied times 36. Locate the 6 on the left hand column of the multiply chart. Move across on that row until you are under the column headed by 5. 6 times 5 for a partial product will be 1E. Put down the E and carry the 1. Multiplying 6 times 4 for the next partial product will give 18. Add the 1 carry to it for a partial product of 19E. Multiply the next digits and add the partial products together for a final product of E8E.

```
 45
 36
---
19E
CF
---
E8E
```

You may find the Addition chart useful during the multiplying when large numbers are to be added.

Multiply the following hexadecimal problems:

```
1.  4335     2.  1002     3.  67F8     4.  1BCF     5.  3ACE
    6389         001A         3FBD         2C61         2DAF
    ----         ----         ----         ----         ----

6.  7FFF     7.  289A     8.  A332     9.  8347    10.  F634
    7FFF         3333         2222         9246         5FFF
    ----         ----         ----         ----         ----
```

DIVIDE:

Divide will possibly be the least used hexadecimal arithmetic. But then, if the dividend of a divide problem were to be checked for accuracy it would be necessary to know how to divide using hexadecimal numbers. Its method, when using the chart, will follow a similar pattern that was used in subtraction. Suppose 6483 is to be divided by 7. Start by locating 7 on the left hand column in the multiplication chart. Move right until you come to a number equal to or less than 64. The number nearest 64 will be 62. The E at the top of that column is the partial quotient. Subtracting 62 from 64 leaves a remainder of 2. Bring down the 8. Move across the 7 row until a number equal to or less than 28 is found. 23 is the nearest number giving a partial product of 5. Continue until the problem is completed.

```
      E5B
 7 ⟌ 6483
     62
      28
      23
       53
       4D
        6
```

Complement negative operands before a divide is started as was done in the multiply. If only the divisor or the dividend is negative, complement the quotent at the end of the divide operation. If both are negative, do not complement the quotent.

Complete the following divide problems:

1. 3 ⟌ 7BCD

2. 6 ⟌ 1463

3. 1A ⟌ 4762

4. 8 ⟌ 7FFF

5. 7 ⟌ 6453

All of the operations which have been gone over could have been converted to binary from the hexadecimal and then completed. The final binary answer would then have to be converted back to hexadecimal. Use the method which is most comfortable for you.

Answers To Practice Problems

Conversion from hex to decimal
Page 24

1. 344,054
2. 65,614
3. 5,742,437
4. 5,667,225
5. 4,759,552
6. -1329
7. -13,569
8. -2,234,107
9. -22,410
10. -163,195,141

Conversion from decimal to hex
Page 26

1. 5E18
2. 12BC8
3. 80000
4. 12D687
5. 747C4A
6. 361DA25
7. 50F16AAD
8. 6FFFFFFF
9. 3B9ACA00
10. C6AEA155

Hex addition
Page 27

1. 5E29
2. C147
3. 3976
4. 1112 an overflow condition exists
5. BE01
6. 0005 notice that FFFE is the same as a -1
7. CDD1
8. 87DD
9. 3E02
10. B64E

Answers to practice problems (cont'd.)

Hex subtraction
Page 27

1. 21EE
2. 1501
3. 8FDB
4. 06FC
5. 8531
6. 1F30
7. 061A
8. 5D63
9. 12EF
10. 21B5

Hex multiplication
Page 28

1. 1A21765D
2. 0001A034 the zeros are inserted to give a 8 digit product found in the 1700
3. 19E2CA18
4. 04D21D6F
5. 0A7E68D2
6. 3FFF0001
7. 081EC4AE
8. F3A072C5
9. 357478F8
10. FC53E9CA the F634 would have to be complemented before the multiplication and then the product would have to be complemented do not complement the product

Hex Division
Page 29

1. 2944 remainder of 1
2. 0365 remainder of 5
3. 02BE remainder of 16
4. 0FFF remainder of 7
5. 0E55 no remainder

CHAPTER I

NON-ADDRESSABLE INSTRUCTIONS

NON-ADDRESSABLE INSTRUCTIONS

INTRODUCTION

Instructions in the 1700 Computer are divided into two major categories -- those which incorporate address modification and those which do not. All addressable instructions have a hexadecimal format of $\boxed{F}\boxed{M}\boxed{-\Delta-}$ where function code F is the most significant hexadecimal digit in the instruction, addressing code M is the second most significant hexadecimal digit in the instruction and the lower half of the instruction forms a modifier or delta (Δ) field.

An instruction is addressable if $F \neq 0$; that is, if its upper-most hexadecimal digit is not a zero. Since there are only 15_{10} addressable instructions in the 1700 instruction repertoire, a single hexadecimal digit (function code F) will suffice to identify the instruction. The addressing code M of an addressable instruction consists of the following bit designation $\boxed{r}\boxed{ind}\boxed{q}\boxed{i}$ where "r" identifies the relative bit, "ind" the indirect bit, "q" the Q Index Register and "i" the Memory Index Register. The significance of these bits and the manner of addressing are explained in another portion of this chapter. The delta field is also explained during the discussion of addressing.

An instruction is non-addressable if $F = 0$. Since all non-addressable instructions must have their upper-most hexadecimal digit equal to 0, this means that additional bits are required to identify the particular instruction. Some non-addressable instructions will require only two hexadecimal digits for identification while others will require more. A general format for all non-addressable instruction is $\boxed{F}\boxed{F_1}\boxed{}$ where $F = 0$ and F_1 identifies the instruction or the group to which an instruction belongs.

Non-addressable instructions may be placed into one of four groups. The register reference group includes those instructions which require only two hexadecimal digits for identification. Most instructions in the group have the format $\boxed{0}\boxed{F_1}\boxed{-\Delta-}$ where the significance of the delta (Δ) field depends on the instruction. To belong to this group of instructions F_1 must not equal 1, 8 or F.

The skip group of instructions are identified by F_1 equaling 1 and have the format $\boxed{0}\boxed{1}\boxed{F_2}\boxed{S}$ where F_2 identifies the particular skip instruction and S represents a skip count. If the skip condition is satisfied, program continuation will be at P+1+S; if not, continuation will be at P+1.

The Inter-register group of instructions are identified by F_1 equaling 8. The format and general operation of this instruction group is given in the 1700 Computer Reference Manual.

Flow diagrams of most 1700 instructions are given on pages A-1 through A-14 in the Appendix. The purpose for such diagrams is to present a simple, overall operation of the instruction (or instruction group) being considered. Some instructions (Multiply or Divide) and the Inter-register instruction group do not lend themselves to simple flow diagrams. For this reason, flow diagrams for these instructions have not been included.

A brief description of each instruction is given in the 1700 Computer Reference Manual, pages 3-2 through 3-20. You should read the appropriate description, then become familiar with the flow diagram as each instruction is being learned. The following pages will provide examples and explanations of the various 1700 instructions.

REGISTER REFERENCE GROUP

Selective Stop (SLS) - | 0 | 0 | ▨ |

Instructions are read out of memory during the Read Next Instruction (RNI) mode. This mode places the instruction into the 1704 X Register and then causes the upper eight bits of the instruction (X) to transfer to the F Register. The contents of the F Register are then translated to determine which instruction or instruction group is to be executed.

Whenever the F Register is all zeros (cleared), the computer executes the Selective Stop (SLS) instruction. There are many ways in which the F Register might become cleared. Each of these possibilities will be explored briefly.

If during RNI a SLS instruction is read from memory, the F Register will become cleared and the computer will execute a SLS instruction. This would be the normal SLS operation. A Master Clear (see page 6-1 of the Reference Manual) will cause the F Register to become cleared. If the computer were first Master cleared before execution of a program, the first operation would be a SLS. Under such a condition the computer bypasses the stop circuitry and prevents stopping. If this were not the case, the computer could never get started following a Master Clear (M.C.) if the Selective Stop switch were on.

The F Register will become cleared if certain illegal operations are attempted. When such operations are encountered, the computer reacts by forcing SLS operations. This prevents execution of the illegal operation and will cause the computer to stop if the Selective Stop switch is on.

Normally, the F Register becomes cleared due to a SLS instruction. For a SLS instruction to stop the computer, the selective stop switch must be on. If the switch is off, the instruction becomes a pass or do-nothing. By definition a pass instruction is one which does only one thing: causes the contents of the P register to be incremented.

For example, suppose the following routine is entered into the computer and executed.

Initial Conditions: M.C., Selective Stop switch on, set P=0100 and press RUN.

0100=0050
0101=1234

Refer to the Selective Stop flow diagram during the following explanation (Appendix, page 1).

Since the computer was master cleared, this clears the F Register so a SLS operation will be executed as soon as the RUN switch is pressed. Beginning at the start of the Selective Stop flow diagram, the following sequence would occur (remember, the first instruction to be executed in the program is at location 0100, which is the present setting of the P Register):

First Pass

1. Is Stop Switch ON? - yes
2. Was instruction preceded by M.C.? - yes
3. Contents of P transfers to the P and Y registers (that is, (P) and (Y) both equal 0100)
4. The next instruction is read from the location specified by the contents of Y. This places 0050 in the X Register, then 00 in the F Register.
5. Is the Stop Circuitry enabled? - no
6. Continue

Second Pass

1. Is Stop Switch ON - yes
2. Was instruction preceded by M.C.? - no, it was preceded by SLS.
3. Enable the Stop Circuitry
4. Increment P and send result to P and Y (this makes (P) and (Y) both equal 0101).
5. The next instruction is read from the location specified by the contents of Y. This places 1234 in the X Register.
6. Computer stops when instruction enters X.

Notice that the contents of the P register is the next consecutive location following the SLS instruction when the computer stops. Also, the X Register contains the contents of the memory location following the SLS instruction.

Enable Interrupt (EIN) — 0 │ 4 │ ////////

The Enable Interrupt (EIN) instruction causes the 1700 Interrupt System to become enabled. Until the interrupt system is enabled no interrupts can occur. Once the interrupt system has been enabled, the computer will be interrupted when one of the selected interrupt conditions occurs.

If the Protect Switch is ON, the EIN instruction can be executed only if it is stored in a protected location (that is, a location having its protect bit a "1"). An attempt to execute an unprotected EIN instruction (with the Protect Switch ON) is considered illegal. The computer will clear the F Register and execute a SLS instruction instead. With the Protect Switch OFF, all instructions are treated as unprotected, thus the EIN instruction would be executed regardless of its protect status.

The computer is so designed that one instruction will be executed after an EIN instruction before the computer can be interrupted. This was included to help simplify interrupt routines. For this course, the reader needs to know only that the earliest an interrupt can occur is during RNI of the 2nd instruction following the EIN instruction which enables the interrupt system.

For example, consider the following routine:

Initial Conditions: M.C., set P=0500, set Mask=0001, Selective Stop switch ON and press RUN

```
0500 --- 0400
0501 --- 0400
0502 --- 0000
0503 --- 0000
```

Since the computer was Master Cleared, the computer will begin with the Selective Stop operation. Upon completion of this pass the contents of (X) = 0400 and (P)=0500. The upper 8 bits of X are transferred to F and translated as an EIN instruction. Following is a sequence of the operations which follow--refer to the Enable Interrupt flow diagram (Appendix, page 2).

Second pass

1. Is the Protect Switch ON? - no (no mention made under initial conditions).
2. Increment contents of P and send result to P and Y (making them equal 0501)
3. Enable the Interrupt System
4. Read next instruction at contents of Y (location 0501. This places the 2nd 0400 instruction in the X register
5. Start third pass

Third pass

1. Upper 8 bits of X transfer to F where the instruction is translated as another EIN.
2. Is the protect switch ON - no
3. Increment (P) and send to P and Y
4. Since interrupt system is already enabled, the instruction is a pass.
5. Read next instruction at location specified by the contents of Y (which is 0502). This places 0000 instruction in the X register
6. Start the fourth pass.

Fourth pass

1. Upper 8 bits of X transfer to F where the instruction is translated as a SLS.
2. Is Selective Stop switch ON? - yes
3. Was instruction preceded by Master Clear? - no
4. Enable the Stop circuitry
5. Increment (P) and send to P and Y
6. Read next instruction at location specified by the contents of Y (location 0503). This places 0000 in the X register.
7. Is stop circuitry enabled? - yes
8. Computer stops.

The start of the fourth pass would have been the first possibility for a computer interrupt. With the Mask register equal to 0001, the only interrupt condition selected is an internal interrupt (see chapter 4 of the Reference Manual). Notice that the second EIN instruction was a do-nothing since the interrupt system was already enabled.

The computer is unable to be interrupted if it is stopped. If an interrupt condition occurs after computer operations stop (even though the interrupt system has been enabled) the condition will be ignored. For example, consider the following program:

Initial Conditions: M.C., set P=0250, set Mask=0001, selective stop switch ON, Internal Interrupt, and press RUN.

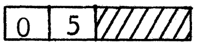
```
0250 --- 0400
0251 --- 0000
0252 --- 0000
```

Due to master clear the first pass will merely cause 0250 to be placed into the Y Register. RNI operations then occur from location 0250 and the instruction 0400 (EIN) is placed in X. During the second pass the EIN instruction is executed and the contents of P incremented making P and Y equal 0251. RNI at 0251 occurs placing the instruction 0000 in X.

During the third pass, the SLS instruction is executed and the contents of P incremented making P and Y equal 0252. RNI operations place the instruction 0000 into X. Since the stop condition was satisfied the computer stops. The internal interrupt condition present will not be recognized since the computer stops before interrupts can be detected. (Interrupt, in this case, would have been detected at the start of the fourth pass.)

One final point concerning the interrupt enable--it remains enabled until one of three conditions occurs:

1. Master Clear
2. Inhibit Interrupt instruction is executed
3. An interrupt occurs.

Inhibit Interrupt (IIN) — 

The Inhibit Interrupt (IIN) instruction causes the 1700 Interrupt System to become disabled. Once disabled, the interrupt system is unable to detect any interruptible conditions.

The IIN instruction requires protection; that is, with the Protect Switch ON the location of the IIN instruction must be protected. An attempt to execute an unprotected IIN instruction (with the protect switch ON) is illegal. Under such a condition, the computer will clear the F Register and execute a SLS instruction instead. With the protect switch OFF, all instructions are treated as being protected.

Consider the following routine:

Initial Conditions: M.C., set P=0710, Selective Stop switch ON, protect switch ON and press RUN - Note, all underlined addresses are protected.

```
0710 --- 0400
0711 --- 0400
0712 --- 0500
0713 --- 0000
0714 --- 0400
```

1. Because of the M.C. condition the first pass will cause the contents of P (0710) to transfer to Y and the instruction (0400) at that location be placed into the X register.

2. The second pass will transfer 04 into the F Register where it is translated as an EIN instruction. Since the protect switch is ON and the location (0710) is protected (underline), the instruction is executed. The contents of P are increased and sent to P and Y and the instruction (0400) at that location (0711) is placed into the X register.

3. The third pass will transfer 04 into the F register where it is translated as an EIN instruction. Since the Interrupt System is already being enabled, this second EIN instruction is a pass. (NOTE: If location 0711 were not protected, the instruction would be illegal and would cause a protect fault, clear the F register, etc.) The contents of P are increased and sent to P and Y and the instruction (0500) at location 0712 is placed into the X register.

4. The fourth pass will transfer 05 to the F register where it is translated as an IIN instruction. Since the protect switch is ON and the location (0712) is protected, the instruction is executed. (NOTE: At the start of this pass interrupts could be detected.) During execution of the IIN instruction, the Interrupt System is disabled and the contents of P are incremented and sent to P and Y. The instruction (0000) at location 0713 is placed into the X Register.

5. The fifth pass will transfer 00 into the F Register where it is translated as a SLS instruction. The contents of P are incremented and the next instruction is read from memory and placed into the X register. Since the conditions for stopping have been satisfied, computer operations stop.

----

1. What would be the status of the 1700 Interrupt System at the time each of the following routines stop?

   a. Initial Conditions: M.C., Selective Stop ON, protect switch ON, set P= 0735 and press RUN

```
0735 --- 0577
0736 --- 0400
0737 --- 0000
0738 --- 0000
0739 --- 0400
```

b. Initial Conditions: M.C., Selective Stop switch ON, set P=1000 and press RUN

```
OFFF=0000
1000=0400
1001=0500
1002=0077
1003=0000
```

2. What will be the contents of the P Register when each of the following routines stop?

a. Initial Conditions: M.C., Selective Stop switch ON, set P=0100 and press RUN

```
0100 --- 0400
0101 --- 0400
0102 --- 00AB
0103 --- 0500
0104 --- 0000
```

b. Initial Conditions: M.C., Selective Stop switch ON, Protect Switch ON, set P=0100 and press RUN

```
0100 --- 0500
0101 --- 0400
0102 --- 0000
0103 --- 0400
0104 --- 0000
```

Set Protect Bit (SPB) - 0 6 ▨

The Set Protect Bit (SPB) instruction is the only means available of making a memory location protected. This instruction requires protection, that is, either the protect switch is OFF or, with the protect switch ON, the storage location of the SPB instruction is protected. An attempt to execute a non-protected SPB instruction with the protect switch ON is illegal resulting in a protect fault, the clearing of the F Register and the execution of a SLS in place of the SPB.

The location to be protected must be placed within the Q Register before execution of the SPB instruction. When executed, the SPB instruction references the location specified by the contents of Q and causes that location to become protected. NOTE: The quantity within that location is unaffected by the SPB instruction.

Consider the following program:

Initial conditions: M.C., set P=0500, selective stop switch ON, set Q=0501 and press RUN

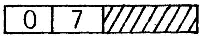
```
0500 --- 0600
0501 --- 0400
0502 --- 0000
0503 --- 0000
```

1. Because of the M.C. condition, the first pass will be a SLS with the contents of P (0500) going to Y. The next instruction is read at the location specified by Y and placed into the X Register.

2. The second pass causes the upper 8 bits of X (06) to transfer to F where it is translated as a SPB instruction. Since the protect switch is OFF, the instruction is legal. Assuming no parity errors, location 0501 will become protected. The contents of P are incremented and sent to P and Y. The next instruction is read at location 0501 (Y) and placed into the X Register.

3. The third pass starts the execution of the EIN instruction, causes P to increment and reads the following instruction into X.

4. The fourth pass executes the SLS instruction (location 0502), enables the stop circuitry, increments P and reads the next instruction into X. The computer stops with the interrupt system enabled and location 0501 protected.

Clear Protect Bit (CPB)— 

The Clear Protect Bit (CPB) instruction is the only means available of clearing the protect bit of a memory location. The location to be cleared is specified by the contents of the Q Register. To be executed, the CPB instruction requires protection. An unprotected CPB instruction (that is, the protect switch ON and the location of the CPB instruction being unprotected) is illegal and results in a protect fault, the clearing of the F Register and the execution of a SLS in place of the CPB.

Consider the following routine:

Initial Conditions: M.C., set P=0300, Selective Stop switch ON, Protect Switch ON, set Q=0301 and press RUN. NOTE: Underlined locations are protected.

```
0300 --- 0600
0301 --- 0700
0302 --- 0000
```

1. The first pass reads the instruction from location 0300 and places into X.

2. The Second pass executes the SPB instruction causing location 0301 to become protected. The next instruction (0700) is read from memory and placed into X.

3. The third pass executes the CPB instruction causing location 0301 to become unprotected. The next instruction (0000) is read from memory and placed into X.

4. The fourth pass executes the SLS instruction causing the stop circuitry to be enabled. The next instruction (contents of 0303) is read from memory and placed into X and the computer stops.

NOTE: The 0700 instruction would have been illegal had not the SPB instruction protected location 0301. This would have caused the computer to stop at the end of the third pass.

Increase A (INA) — | 0 | 9 | —△— |

The Increase A (INA) instruction provides a means of modifying the contents of the A Register without the need of storing the amount to be modified in memory, then using an ADD or SUB instruction. This, obviously, saves time since memory does not have to be referenced to execute the instruction. Use of the instruction is limited, however, to those applications where the change is no greater than ± 7F.

A 09 code in the F Register identifies the INA instruction. The quantity △ (lower half of the instruction) is treated as a signed quantity (the upper bit of △ being the sign bit). The sign of this quantity is extended throughout X then added to the contents of A, the result going to A.

For example, the instruction 0954 would cause 0054 to be added to the contents of A while the instruction 09A7 would cause FFA7 to be added to the contents of A. Since an arithmetic operation is involved, overflow status is recorded by the computer. Overflow indicates that the result lies outside the range of the computer. One of the skip instructions can be used by the programmer to determine whether or not overflow occurred.

Consider the following routine:

Initial condition: M.C., set P=0179, set A=037A, Selective Stop switch ON and press RUN.

```
0179 --- 0901
017A --- 0000
```

1. Because of the M.C., the first pass would read the instruction at location 0179 and place into the X Register.

2. The INA instruction is executed causing the quantity 0001 to be added to the contents of A (037A) resulting in 037B in the A Register. The next instruction is read from memory and placed into the X Register.

3. The SLS instruction is executed causing the stop circuitry to be enabled. The next instruction is read into X and the computer stops.

NOTE: If the contents of location 0179 had been 09FE, the quantity FFFE (-1) would have been added to the A Register, causing the contents of A to be reduced by 1.

ENTER A (ENA) - $\boxed{0 \mid A \mid -\Delta-}$

The Enter A (ENA) instruction provides a means of placing small quantities into the A Register without the need of previously storing that quantity in memory, then using a Load A (LDA) instruction to place the quantity in the A Register. Use of the instruction is limited, however, to those applications where the contents of A are no greater than ± 7F.

A 0A code in the F Register identifies the ENA instruction. The quantity △ (lower half of the X Register) is treated as a signed quantity (the upper bit of being the sign bit). The sign of this quantity is extended throughout X then transferred to the A Register.

Consider the following routine:

Initial Conditions: M.C., Set P = 0200, Set A = 247A, Selective Stop switch ON and press RUN.

```
0200 --- 0A85
0201 --- 0000
```

1. The ENA instruction causes FF85 to be placed in the A Register.

2. The SLS instruction stops the computer after reading the next instruction into the X Register.

   NOTE: Had the contents of 0200 been 0A75, the quantity 0075 would have been placed in the A Register.

NO OPERATION - $\boxed{0 \mid B \mid ▨▨▨}$

A 0B code in the F Register identifies the No Operation instruction. This instruction is a pass or do-nothing operation. In writing machine language programs, it is sometimes wise to include a No Operation instruction every so often. Since such instructions can be inserted or removed without affecting the operation of the program, they provide locations within the program for absorbing changes. For example: Suppose a program is written with every 5th instruction a No Operation type. Later, the program is revised by inserting two instructions previously omitted. The two instructions would be inserted in the proper sequence then two subsequent No Operations removed, localizing the program change to an area of ten instructions or so. If the original program had been written without No Operations, insertion of additional instructions would cause <u>all</u> subsequent instruction locations to change.

Another use of the No Operation instruction is to replace an instruction deleted during program revision. This involves minimum effort on the part of the programmer since only one location changes for each instruction deletion.

Once a program has been tried and proven, the programmer can then rewrite the entire program removing all No Operations. His first concern is to get a program written as soon as possible that works; then he concerns himself with making the program efficient (if efficiency is important).

Consider the following routine:

Initial Conditions: M.C., set P=0100, Selective Stop switch ON, protect switch ON
set Q=0100 and press RUN

```
0100 --- 0600
0101 --- 0400
0102 --- 0901
0103 --- 0400
0104 --- 0000
```

The preceding program could be rewritten in the following manner without affecting operation.

```
0100 --- 0B00
0101 --- 0400
0102 --- 0901
0103 --- 0B00
0104 --- 0000
```

- - - - - - -

1. Give the final contents of A and P when each of the following routines stop.

   a. Initial Conditions: M.C., Set P=0200, Set Q=0201, Selective Stop switch ON, Protect switch ON and press RUN.

```
0200 --- 0600
0201 --- 0400
0202 --- 09FE
0203 --- 0700
0204 --- 0902
0205 --- 0000
```

   b. Initial Conditions: M.C., set P=0700, set Q=0702, set A=077A, Selective Stop switch ON, and press RUN.

```
0700 --- 0600
0701 --- 0400
0702 --- 0A05
0703 --- 0500
0704 --- 090B
0705 --- 0B00
0706 --- 0000
```

c. Initial Conditions: M.C., set P=09AF, set A=F0F0, selective stop switch ON, protect switch ON and press RUN.

```
09AF --- 09FE
09B0 --- 0700
09B1 --- 0A00
09B2 --- 0600
09B3 --- 0B00
09B4 --- 0075
09B5 --- 0000
```

2. What is the status of the interrupt system after each of the following programs?

a. Initial conditions: M.C., Selective Stop switch ON, protect switch ON, set Q=0002 and press RUN.

```
0000 --- 0700
0001 --- 0400
0002 --- 0500
0003 --- 0600
0004 --- 0000
```

b. Initial Conditions: M.C., set P=0050, set Q=0052, Selective Stop switch ON, Protect Switch ON and press RUN

```
0050 --- 0B00
0051 --- 0600
0052 --- 0400
0053 --- 0955
0054 --- 0B00
0055 --- 0700
0056 --- 0000
```

ENTER Q (ENQ) — $\boxed{0}\boxed{C}\boxed{-\triangle-}$

The Enter Q (ENQ) instruction provides a means of placing small quantities into the Q Register without the need of having previously stored those quantities in memory then using a load Q (LDQ) instruction to place them in the Q Register. Use of the instruction is limited, however, to those applications where the quantity in Q is no greater than ± 7F.

A OC Code in the F Register identifies the ENQ instruction. The quantity △ (lower half of the instruction) is treated as a signed quantity. While in the X Register, △ undergoes sign extension. The resultant 16-bit quantity is then transferred to the Q Register.

Consider the following routine:

Initial Conditions: M.C., Set P=0259, set Q=07F4, Selective Stop switch ON and press RUN

```
0259 --- 0C00
025A --- 0000
```

1. The ENQ instruction places 0000 in the Q Register.

2. The SLS instruction stops the computer after reading the next instruction into the X Register.

NOTE: Had the contents of location 0259 been OCFF, the quantity FFFF would have been entered in the Q Register.

Increase Q (INQ) — $\boxed{0}\boxed{D}\boxed{-\triangle-}$

The Increase Q (INQ) instruction provides a means of modifying the contents of the Q Register within the range of ±7F without the need of referencing memory. A OD code in the 1704 F Register identifies the INQ instruction. The quantity △ (lower half of the instruction) is treated as a signed quantity. While in the X Register, △ undergoes sign extension. The contents of Q are then added to the 16-bit quantity in X with the sum going to Q. Since an arithmetic operation is involved, overflow status is recorded by the computer.

Consider the following routine:

Initial conditions: M.C., set P=1000, set Q=00FF, selective stop switch on and press RUN

```
1000 --- 0D05
1001 --- 0000
```

1. The INQ instruction will form the sum of 0005 (△ with sign extension) plus 00FF (contents of Q), resulting in 0104 in the Q Register.

2. The SLS instruction will stop the computer after reading the next location (1002) into the X Register.

NOTE: If the contents of location 1000 had been ODFA, the final contents of Q would have been FFFA plus OOFF, resulting in OOFA, that is, 5 would have been subtracted from A (FFFA is a-5).

Exit Interrupt (EXI) —

0	E	—Δ—

The Exit Interrupt (EXI) instruction will always be the last instruction of an interrupt subroutine and provides for automatic return to the operation which was interrupted. When a computer interrupt occurs, the operation being performed is temporarily set aside and a routine to process the interrupt is performed. At the time of interrupt, the computer stores into memory the address to which it must return after completing the interrupt routine. Since there are 16 possible interrupt levels (and one level can interrupt another level), 16 locations are set aside in memory where the return address will be stored -- each level having its own location. These 16 locations begin at address 0100 and differ by a hexadecimal count of 4 so that level 0 would have its return address stored at location 0100, level 1 would have its return address stored at location 0104, level 2 at location 0108, level 3 at 010C, level 4 at 0110, etc. Table 4-1 in the 1700 Reference Manual lists the location for all interrupt levels.

Notice that all return address locations have the same upper two hexadecimal digits (01). The EXI instruction will identify the lower two digits of the return address location via the delta field. Thus, if the EXI instruction appears in the level 0 subroutine, it will be written 0E00, if it appears in level 1, it will be written 0E04, etc.

To be executed, the EXI instruction must be protected; that is, either the protect switch is OFF or the location of the EXI instruction is protected. When executed, the computer forms the address 01 Δ in the Y register, which should be the location of the return address for that interrupt level. The return address is read from memory and placed into X. The 16th bit of this quantity represents the status of overflow at the time interrupt occurred. This status is automatically returned to the computer overflow circuitry during the EXI instruction. The lower 15 bits of X represent the return address and are forced into P and Y. The interrupt system is automatically enabled (thereby eliminating the need for EIN instructions to re-enable the interrupt system) and the computer continues by reading the instruction at the location specified by Y.

If the EXI instruction is not protected, an attempt to execute it is illegal resulting in a protect fault, the clearing of the F Register and the execution of a SLS instruction in place of the EXI.

NOTE: Having the wrong digits in the Δ field of the EXI instruction is a programming error for which the computer is not responsible. Execution of the EXI instruction will always read location 01 Δ as a return address, will always transfer bit 16 to the overflow circuitry (a negative quantity would indicate overflow) and use the lower 15 bits as a return address.

Consider the following routine:

Initial Conditions: M.C., set P=0500, Selective Stop switch ON, Protect Switch ON and press RUN

```
0100 --- 9005
0500 --- 0E00
0501 --- 0000
1005 --- 0000
```

1. The EXI instruction (location 0500) reads location 0100 and places the quantity 9005 in X. The upper bit of X (a "1") is sent to the overflow circuitry (indicating, in this case, an overflow). The interrupt system is enabled and the quantity 1005 (lower 15 bits of X) is sent to P while Y receives all 16 bits (9005). The next instruction is read from the location specified by the lower 15 bits of Y (1005).

2. The SLS instruction at location 1005 causes the computer to stop after reading the next instruction into the X Register.

- - - - - -

1. What would be the final contents of P, A and Q in each of the following routines?

   a. Initial Conditions: M.C., set P=0150, Set A=1000, Set Q=F754, Selective Stop switch ON, protect switch ON and press RUN.

```
0150 --- 0400
0151 --- 0C7F
0152 --- 0D7F
0153 --- 0D57
0154 --- 0600
0155 --- 0E50
0156 --- 0A00
0157 --- 0000
0400 --- 0000
```

   b. Initial Conditions: M.C., set P=0100, Selective Stop switch ON and press RUN

```
0100 --- 0600
0101 --- 0901
0102 --- 0E00
0103 --- 0D01
0104 --- 0105
0105 --- 0000
0600 --- 0CFF
0601 --- 0E04
0602 --- 0000
```

2. What will the following routine do?

Initial Conditions: M.C., set P=00FF and press RUN

```
00FF --- 0600
0100 --- 00FF
0101 --- 0D01
0102 --- 0E00
- - - - - - - - - - - - -
```

SKIP GROUP

Instructions belonging to the Skip group are identified by the upper two digits being 01. Individual skip instructions are identified by the third (F2) digit. In each case, a skip instruction senses a given condition within the computer. If the condition being sensed exists, the skip is satisfied and program execution continues at (P) +S+1, where S is the lowest digit of the skip instruction (always positive). If the condition being sensed does not exist, the skip is not satisfied and program execution continues at (P) + 1.

SKIP IF (A) = +0 (SAZ) — $\boxed{0\,|\,1\,|\,0\,|\,S}$

When the SAZ instruction is executed, the entire contents of the A Register are checked. If the contents are found to be all zeros (+0), the skip is satisfied. Any bit in the A Register being a "1" will disable the skip instruction, converting it to a pass.

The SAZ instruction might be used to terminate a counting operation, such as in the following routine.

Initial Conditions: M.C., set P=0170, set Q=1000, set A=0100, Selective Stop switch ON and press RUN

```
016F --- 0170
0170 --- 09FE
0171 --- 0103
0172 --- 0600
0173 --- 0D01
0174 --- 0E6F
0175 --- 0000
```

The preceding routine protects successive memory locations starting at the location specified by the initial contents of Q. The number of locations to be protected is determined by the initial contents of A. In this particular routine, locations 1000 through 10FF become protected after which time the computer stops.

SKIP IF (A) ≠ + 0 (SAN) - $\boxed{0\,|\,1\,|\,1\,|\,S}$

When the SAN instruction is executed, the skip condition becomes satisfied if any bit in the A Register is a "1". This instruction might be used to terminate a counting operation, such as in the following routine.

Initial Conditions: M.C., set P=0110, set Q=1000 set A=0100, Selective Stop switch ON and press RUN

```
010F     0110
0110 --- 09FE
0111 --- 0111
0112 --- 0000
0113 --- 0600
0114 --- 0D01
0115 --- 0E0F
```

This routine accomplishes the same thing as that given previously for the SAZ instruction; that is, locations 1000 through 10FF are protected. Notice that skipping occurs every pass through the routine except when the contents of A have been reduced to +0.

SKIP IF (A) IS + (SAP) - $\boxed{0 \mid 1 \mid 2 \mid S}$

When the SAP instruction is executed, the sign bit of the A Register is sensed. If the bit is a "0" (meaning A is positive), the SAP instruction will result in a skip. If, instead, the quantity in the A Register is negative, the SAP instruction becomes a pass.

SKIP IF (A) IS - (SAM) - $\boxed{0 \mid 1 \mid 3 \mid S}$

When the SAM instruction is executed, skipping occurs if the quantity in A is negative; otherwise, the instruction becomes a pass.

SKIP IF (Q) = +0 (SQZ) - $\boxed{0 \mid 1 \mid 4 \mid S}$

The SQZ instruction is the same as the SAZ instruction except that the contents of the Q Register are being sensed.

SKIP IF (Q) ≠ +0 (SQN) - $\boxed{0 \mid 1 \mid 5 \mid S}$

The SQN instruction is the same as the SAN instruction except that the contents of the Q Register are being sensed.

SKIP IF (Q) is + (SQP) - $\boxed{0 \mid 1 \mid 6 \mid S}$

The SQP instruction is the same as the SAP instruction except that the sign of the Q Register is being sensed.

SKIP IF (Q) IS - (SQM) - $\boxed{0 \mid 1 \mid 7 \mid S}$

The SQM instruction is the same as the SAM instruction except that the sign of the Q Register is being sensed.

SKIP IF SKIP SWITCH SET (SWS) - $\boxed{0 \mid 1 \mid 8 \mid S}$

The SWS instruction monitors the status of the Skip switch on the 1700 console. If the switch is set (ON), the SWS instruction will result in a skip; if the switch is not set (OFF), the SWS instruction becomes a pass.

SKIP IF SKIP SWITCH NOT SET (SWN) -

0	1	9	S

The SWN instruction results in a skip if the console skip switch is not set (OFF). If the switch is set, the SWN instruction becomes a pass.

SKIP IF OVERFLOW (SØV) -

0	1	A	S

The SØV instruction senses the status of the overflow circuitry. If an Overflow condition is sensed, the SØV instruction will result in a skip. Overflow can occur as the result of any arithmetic operation except multiply. Once overflow occurs, the overflow condition remains until the computer is Master Cleared or a Skip instruction which senses the overflow status is executed. The SØV instruction, then, always leaves the computer in a no overflow state regardless whether skipping occurred or not.

Consider the following routine:

Initial Conditions: M.C., set P=0200, set A=7FFF, Selective Stop switch set and press RUN

```
0200 --- 0901
0201 --- 01A1
0202 --- 0000
0203 --- 0000
```

In the preceding routine, the INA instruction results in an overflow so that when the SØV instruction is executed a skip will result and the overflow condition will be removed.

SKIP IF NO OVERFLOW (SNØ) —

0	1	B	S

The SNØ instruction will result in a skip if an overflow condition is not present. If an overflow condition is present, the SNØ instruction becomes a pass. After sensing the overflow status, the SNØ instruction leaves the computer in a no overflow state regardless whether skipping occurred or not.

SKIP IF PARITY ERROR (SPE) —

0	1	C	S

The SPE instruction senses the status of the Parity error circuitry within the computer. Whenever memory is referenced, the quantity at the location being referenced is checked for parity. In the 1700 computer, parity is odd; that is, the total number of "1" bits in any memory location (including the protect and parity bits) must be odd. When quantities are stored in memory odd parity is insured by forcing the parity bit to be the proper configuration. Whenever a memory location is referenced, then, total parity should be odd. If it is not, it indicates a parity error. Once a parity error condition occurs it remains until the computer is master cleared or an instruction which senses parity error status is executed.

The SPE instruction will result in a skip if a parity error exists at the time the instruction is executed and will become a pass if no parity error exists. In either case the SPE instruction leaves the computer in a no parity error state.

SKIP IF NO PARITY ERROR (SNP) —

0	1	D	S

The SNP instruction will result in a skip if no parity error is present at the time the instruction is executed. If there is a parity error, the SNP instruction becomes a pass. After sensing the parity error status, the SNP instruction leaves the computer in a no parity error state.

SKIP IF PROTECT FAULT (SPF) —

0	1	E	S

The SPF instruction senses the status of the protect fault circuitry within the computer. A protect fault will occur under one of three conditions:

1. An illegal store into memory
2. An illegal execution of EIN, IIN, SPB, CPB, EXI instructions or any interregister instruction having the Mask Register as a destination.
3. An attempt to execute a protected instruction after a non-protected one.

An illegal store occurs whenever a non-protected instruction attempts to store a quantity into a protected memory location (with the protect switch ON).

When a protect fault occurs, the fault condition persists until the computer is Master Cleared or an instruction is executed that senses protect fault status.

The SPF instruction will result in a skip if a protect fault exists at the time the instruction is executed. If no protect fault exists, the SPF instruction becomes a pass. In either case, the instruction will leave the computer in a no protect fault state.

Consider the following routine:

Initial Conditions: M.C., set P=1000, Selective Stop switch ON, Protect
                    Switch ON and press RUN

```
1000 --- 0400
1001 --- 0901
1002 --- 0B00
1003 --- 01E1
1004 --- 0000
1005 --- 0000
```

The preceding routine causes the Interrupt System to become enabled (EIN instruction). The contents of A are then increased making A=0001. The third instruction is a pass. When this instruction is read from memory, however, a protect fault occurs (protected following non-protected) which converts the instruction to a SLS. The computer stops after reading the SPF instruction into X.

Note: If, after the first stop, the computer were restarted without first Master Clearing, the SPF instruction would cause a skip to location 1005 and clear the protect fault condition.

SKIP IF NO PROTECT FAULT (SNF) —

0	1	F	S

The SNF instruction will result in a skip if no protect fault is present at the time the instruction is executed. If there is a protect fault, the SNF instruction becomes a pass. In either case, the SNF instruction leaves the computer in a no protect fault state.

- - - - - - - - - - - - - -

What are the final contents of A, Q and P after each of the following routines?

1. Initial Conditions: M.C., set P=0500, Selective Stop switch ON, set Q=800F and press RUN

```
015F --- 0160
0160 --- 01A1
0161 --- 0000
0162 --- 0000
0500 --- 0DEF
0501 --- 0400
0502 --- 0E5F
0503 --- 0000
```

2. Initial Conditions: M.C., set P=0200, Selective Stop switch ON, set Q=0202 and press RUN

```
00FF     0B00
0100 --- 00FF

0200 --- 0600
0201 --- 0400
0202 --- 0E00
0203 --- 01F1
0204 --- 0000
0205 --- 0000
```

3. Initial Conditions: M.C., set P=0101, Set A=0001, Selective Stop switch ON and press RUN

```
0100 --- 0101
0101 --- 0132
0102 --- 0155
0103 --- 0000
0104 --- 0901
0105 --- 0141
0106 --- 0E00
0107 --- 0000
0108 --- 09FE
0109 --- 0141
010A --- 0E00
010B --- 0000
```

4. What would have been the final contents of A, Q and P in the routine for problem #3 if the initial contents of A had been FFFE?

5. Under what condition will the following routine stop?

Initial conditions: M.C., set P=0150, Selective Stop switch ON and press RUN

```
014F --- 0150
0150 --- 0183
0151 --- 0901
0152 --- 0DFE
0153 --- 0E4F
0154 --- 0000
```

6. In the routine for problem #5, what relationship will the contents of A and Q have to each other each time the computer is stopped?

INTERREGISTER (IR) Group

Instructions belonging to the Interregister group are identified by the upper two digits being 08. Individual IR instructions are identified by the lower two digits, which have the following format:

7	6	5	4	3	2	1	0
LP	XR	A	Q	M	A	Q	M
Operation		Source			Destination		

Bits 6 and 7 of the IR instructions determine the operation to be performed and have the following significance:

1. LP=0, XR=0 indicates Add operation (symbol +)
2. LP=0, XR=1 indicates Exclusive OR operation (symbol $\veebar$)
3. LP=1, XR=0 indicates Logical Product operation (symbol $\wedge$)
4. LP=1, XR=1 indicates Complemented Logical Product operation (symbol $\overline{\wedge}$)

Bits 3, 4 and 5 determine the operand source upon which the preceding operations may be performed. Two operands are involved with every IR instruction. One operand is determined by the status of bit 5. If the "A" source bit is a "1", the first operand will be the contents of the A Register. If the "A" source bit is a "0", the first operand will be negative zero (the quantity FFFF). The second operand is determined by the status of both bits 3 and 4.

If both bits 3 and 4 are zeros, the second operand is negative zero (FFFF); if the Q source bit is a zero and the M source bit is a "1", the second operand is the contents of the Mask Register; if Q source= "1" and M source = "0", the second operand is the contents of the Q Register; if Q source = "1" and M source = "1", the second operand is the inclusive OR (symbol $\vee$) of the Q Register and Mask Register contents.

Don't get the inclusive OR and exclusive OR operations confused. If an IR instruction had the follow bit configuration

7	6	5	4	3	2	1	0
0	1	1	1	1			

it would mean that an exclusive OR (symbol $\forall$) of operands 1 and 2 is to be performed where operand 1 is the contents of the A Register and operand 2 is the inclusive OR of the Q and Mask Register contents. This would be written as [A] $\forall$ [(Q) V (M)]

On the other hand, an IR instruction having the configuration

7	6	5	4	3	2	1	0
1	0	1	1	1			

would mean that a logical product (symbol $\wedge$) of operands 1 and 2 is to be performed where operand 1 is the contents of the A Register and operand 2 is the inclusive OR of the Q and Mask Register contents. This would be written as [A] $\wedge$ [(Q) V (M)]

Bits 0, 1 and 2 of the IR instructions determine the destination registers which are to receive the result of the operation specified. If a destination bit is a "1", the corresponding register receives the result. Any combination of destination registers may be indicated so that the A, Q and Mask registers can all receive the same quantity. If no destination bits are ones, the operation is performed but the result is lost. If the operation being performed is an add (bits 6 and 7 both zeros), overflow status is recorded. With an add operation specified and no destination bits set, a programmer can perform magnitude comparisons without destroying the operands. This enables overflow situations to be predicted and corrective action to be taken (if necessary) in advance of an addition.

Two features of the IR instruction operations bears further consideration. If one operand is negative zero and a logical product is performed, the result is that the other operand transfers directly to the destination registers. For example if an IR instruction has the following format,

7	6	5	4	3	2	1	0
1	0	0	1	0	1	0	0

the contents of the Q Register would transfer to the A Register. The following format would cause the contents of the A Register to transfer to the Q and Mask Registers.

7	6	5	4	3	2	1	0
1	0	1	0	0	0	1	1

If one operand is negative zero and an exclusive OR is performed, the result is that the other operand becomes complemented into the destination register. Thus, to complement the contents of the A Register would require an IR instruction having the following format:

7	6	5	4	3	2	1	0
0	1	1	0	0	1	0	0

One final precaution concerning the IR instructions. Any IR instruction specifying the Mask register as a destination must be protected (The protect switch OFF or the location of the instruction be protected) or the instruction is considered illegal. If an illegal IR instruction is attempted, a protect fault occurs and a SLS instruction will be executed in place of the IR instruction.

> NOTE: If the lowest hexadecimal digit of the IR instruction is odd, the Mask destination bit is a "1".

- - - - - - - - - - -

1. Show in hexadecimal format an IR instruction which will:

   a. $[A] + [Q] \longrightarrow A$

   b. $\overline{[(Q) \vee (M)]} \longrightarrow A$

   c. $[A] \ \overline{\wedge} \ [(Q) \vee (M)] \longrightarrow$ Q and M

   d. $\overline{[A]} \longrightarrow$ A, Q and Mask

Table VII is included to enable analysis of logical operations to be performed in hexadecimal rather than binary, thereby saving time. An example in the use of table VII

Suppose an inter - register instruction is being executed that performs:

$$[A] \;\; \forall \;\; [(Q) \;\; \vee \;\; (M)]$$

Given: (A) = 15F3, (Q) = 74B2 and (Mask) = 0F05, the result would be:

1. Form the inclusive OR ($\vee$) of the Q and Mask Register contents using the appropriate portion of table VII; thus:

$$\begin{array}{rcl} (Q) & = & \text{74B2} \\ (M) & = & \underline{\text{0F05}} \\ (Q) \vee (M) & = & \text{7FB7} \end{array}$$

2. Form the exclusive OR ($\forall$) of the A Register and the result obtained in step 1 using the appropriate portion of table VII; thus:

$$\begin{array}{rcl} (A) & = & \text{15F3} \\ (Q) \vee (M) & = & \underline{\text{7FB7}} \\ [A] \forall [(Q) \vee (M)] & = & \text{6A44} \end{array}$$

2. What would be the final contents of A, Q and Mask registers upon completion of the following routines?

   a. Initial Conditions: M.C., Set P = 0500, Set A = 0050, Set Q = FFF0, Selective Stop switch ON and press RUN

```
0500 --- 0830
0501 --- 01A6
0502 --- 0851
0503 --- 0834
0504 --- 01B1
0505 --- 0900
0506 --- 0891
0507 --- 0000
0508 --- 0C00
0509 --- 0817
050A --- 0000
```

   b. Initial Conditions: M.C., Set P = 0170, Selective Stop switch ON, Protect switch ON and press RUN

```
0170 --- 0AFF
0171 --- 08A1
0172 --- 0977
0173 --- 0837
0174 --- 0000
```

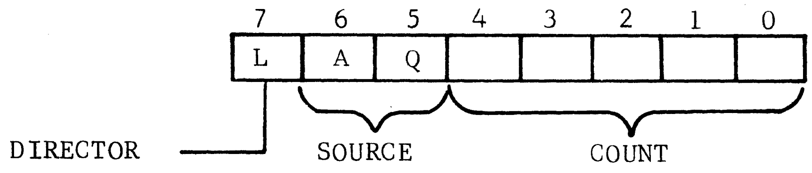
   c. Initial Conditions: M.C., Set P = 0191, Set A = 1234, Set Q = 7000, Selective Stop switch ON, Protect switch ON and press RUN

```
0191 --- 0834
0192 --- 01B1
0193 --- 0191
0194 --- 0E93
0195 --- 0600
0196 --- 0851
0197 --- 0181
0198 --- 0000
0199 --- 0E93
```

- - - - - - - -

SHIFT GROUP

Instructions belonging to the Shift group are identified by the upper two digits being OF. Individual shift instructions are identified by bits in the lower half of the instruction, which have the following format:

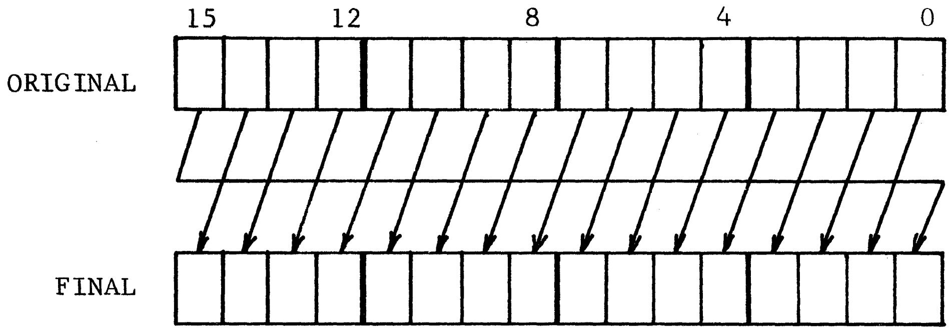


Bit 7 is a director bit which determines the direction in which shifting will occur. If L = "1", shifting is to the left whereas if L = "0", shifting is to the right.

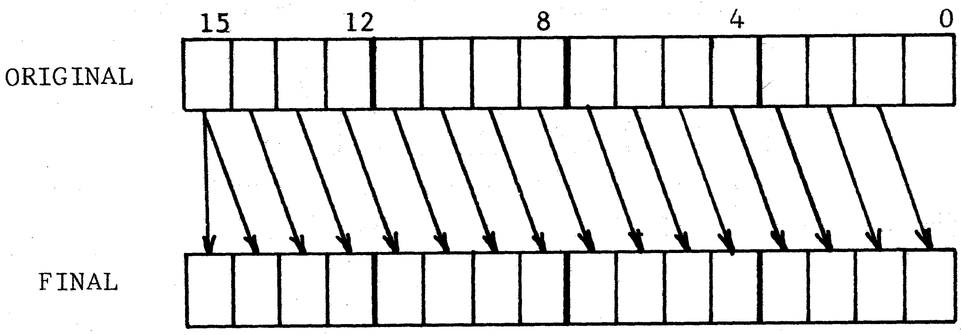
Bits 5 and 6 identify the source of the shifted quantity. If A = "1" the A Register contents will be shifted; if Q = "1" the Q Register contents will be shifted; if both A and Q bits = ones, both register contents will be shifted and if both A and Q bits = zeros, no register contents will be shifted.

Bits 0 through 4 identify the shift count. As shifting progresses, the shift count is reduced each time the source is shifted one position. Shifting continues until the shift count is reduced to zero.

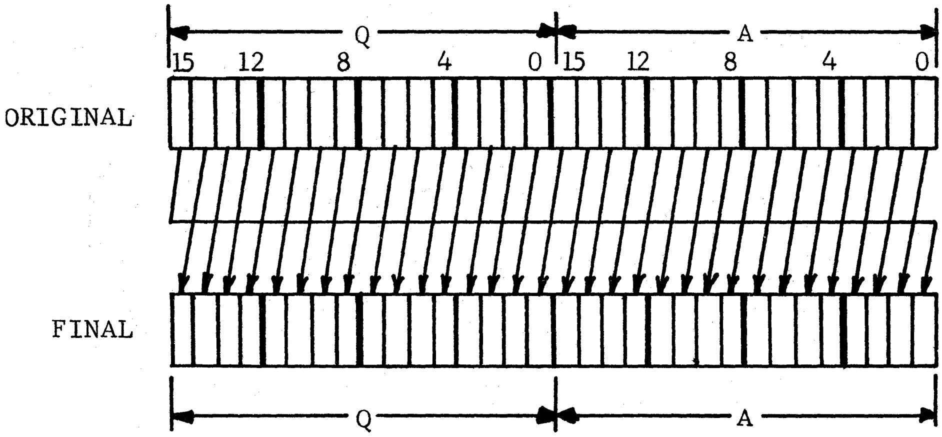
If the source is a single register, left shifting will be end-around such as:



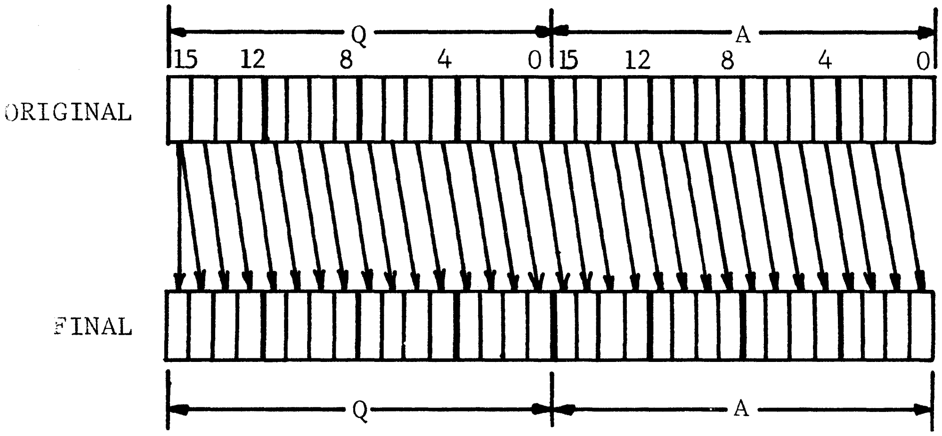
while right shifting will be end-off with sign extension such as:



If the source is both A and Q, the registers will be treated as a single 32-bit QA register so that left shifting will be end around as:



while right-shifting will be end-off with sign extension such as:



- - - - - - - - -

1. What is the final contents of A and Q in each of the following routines?

   a. Initial Conditions: M.C., Set P = 1000, Set A = 1234, Set Q = FE03, Selective Stop switch ON and press RUN

```
1000 --- 0F24
1001 --- 0FF0
1002 --- 0000
```

   b. Initial Conditions: M.C., Set P = 1500, Set A = 0FF1, Set Q = 0185, Selective Stop switch ON and press RUN

```
1500 --- 0F85
1501 --- 0F5F
1502 --- 0CFF
1503 --- 0F68
1504 --- 0000
```

2. Write a routine that will take the original contents of the A register and reverse the bits in every pair; that is, bits 0 and 1 reverse, bits 2 and 3 reverse, bits 4 and 5 reverse, etc.

3. Write a routine that will count the number of "1" bits in the A register and place that count in the Mask register.

4. Write a routine that will require one second (accurate within a few microseconds) to be executed.

5. Write a routine that will cause the overflow indicator to turn on and off for one-second alternations repeated 10 times.

- - - - - - - - - - -

ANSWERS -- CHAPTER I

Pages 1-7 & 1-8

1. a. Disabled. The instruction at location 0736 is illegal and causes the computer to stop. The only instruction executed is the IIN instruction at location 0735.

   b. Disabled. Protect status isn't monitored. The IIN instruction at location 1001 overcomes the effects of the EIN instruction at location 1000. NOTE: The program starts at location 1000, not OFFF.

2. a. (P) = 0103. Always advance (P) and read up the next instruction before stopping.

   b. (P) = 0101. The instruction at location 0100 is illegal and causes the computer to stop.

Pages 1-12 & 1-13

1. a. (A) = FFFE, (P) = 0204

   b. (A) = 0010, (P) = 0707

   c. (A) = 0000, (P) = 09B3

2. a. The 0500 instruction at location 0002 is illegal and stops computer with (P) = 0003.

   b. Enabled. All instructions are executed until the CPB instruction (location 0055) is attempted. This instruction is illegal and stops the computer with (P) = 0056 and (A) = 0055.

Pages 1-16 & 1-17

1. a. (P) = 0401, (A) = 1000, (Q) = 0155

   b. (P) = 0106, (A) = 0001, (Q) = FFFF

2. Will cause all memory locations to become protected. By stepping through the routine, specific locations can be protected by manually setting Q to the location to be protected. Each pass through the routine will protect one location and automatically increment Q. The routine stops when the Selective Stop switch is turned on.

ANSWERS (continued)

Pages 1-21 & 1-22

1. (A) = 0000, (Q) = 7FFF, (P) = 0163

2. (A) = 0000, (Q) = 0202, (P) = 0101

3. (A) = 0001, (Q) = 0000, (P) = 0104

4. (A) = 0000, (Q) = 0000, (P) = 0108

5. The routine will stop when the Skip switch is turned ON.

6. The two registers will always contain complements of each other's contents.

Pages 1-24 & 1-25

1. a. 0834

   b. 085C or 08DC

   c. 08FB

   d. 0867 or 08E7

2. a. (A) = 0041, (Q) = FFF0, (Mask) = FFF0

   b. (A) = FFFF, (Q) = 0000, (Mask) = 0000

   NOTE: The unprotected instruction at location 0171 is a transfer to the Mask Register. With the Protect Switch ON, the instruction is illegal causing the computer to stop.

   c. (A) = 8234, (Q) = 7000, (Mask) = 0000

   NOTE: The instruction at location 0191 causes an overflow. The instruction at location 0195 is illegal and stops the computer.

ANSWERS (continued)

Pages 1-28 & 1-29

1. a. (A) = FFE0, (Q) = 1234

   b. (A) = FF00, (Q) = FFFF

2. Routine will be checked by running it on a computer

3. same as #2

4. same as #2

5. same as #2

CHAPTER II

ADDRESSABLE INSTRUCTIONS

ADDRESSABLE INSTRUCTIONS

INTRODUCTION

Any instruction whose uppermost digit is non-zero is an addressable (or Storage Reference) instruction. These instructions are characterized by two facts; they all have memory reference capabilities and incorporate addressing modes to arrive at the proper memory location to be referenced.

The format of all addressable instructions is

F	M	—△—

where F identifies the instruction, M identifies the addressing code and the delta (△) field is a modifier. The addressing code M of the instruction consists of four bits having the following format:

11	10	9	8
r	ind	q	i

where the "r" bit indicates relative, the "ind" bit indicates indirect, the "Q" bit indicates Q Register indexing and the "i" bit indicates Memory Register indexing. The significance of these addressing bits and the manner in which addressing is accomplished will be explained in a latter portion of this chapter.

INSTRUCTIONS

Jump (JMP) -

1	M	—△—

The Jump (JMP) instruction provides a means of leaving the normal instruction sequence and continuing program execution from some other portion of memory (called branching). Program continuation will be at the effective address (determined by addressing). For example, assume a JMP instruction is located at address 1000 and the effective address is 1500, program sequence would go from the JMP instruction at location 1000 to the instruction at location 1500 as if the two instructions were in consecutive memory locations.

One important use of the Jump instruction occurs when the effective address forces branching to a previous instruction. This provides a means of looping back to re-executed instructions common to a given operation. The programmer would include a counting routine which would determine the number of passes through the loop.

Multiply Integer (MUI) -- $\boxed{2}\boxed{M}\boxed{\text{---}\Delta\text{---}}$

The Multiply Integer (MUI) instruction forms a 32-bit product of the contents of the A Register and the contents of the memory location specified by the effective address. The contents of A is the multiplier while the contents of the effective address is the multiplicand. The 32-bit product will appear in the Q and A registers, Q containing the significant half. The original contents of A and Q become destroyed as a result of the MUI instruction.

Since the 1700 uses one's complement arithmetic, a negative number is merely the complement of a positive number. This introduces the negative zero (FFFF) quantity, since it is the complement of positive zero (0000). There is no such concept as -0 in arithmetic and whenever such a result is obtained from a computer, a problem exists. Obtaining a -0 result due to arithmetic operations can be minimized by adder design; however, certain combinations of operands will still produce a -0 result. For the 1700, MUI instructions will produce a negative zero only under the following conditions:

(+0) x (-N)
(-N) x (+0)
(-0) x (+N)
(+N) x (-0)

Overflow can never occur as the result of a multiplication since the product of the two largest possible operands does not exceed the 32-bit capacity available. (The reader might prove this to himself by squaring 7FFF).

Divide Integer (DVI) -- $\boxed{3}\boxed{M}\boxed{\text{---}\Delta\text{---}}$

The Divide Integer (DVI) instruction takes the 32-bit dividend in QA and divides by the contents of the memory location specified by the effective address. The original contents of Q is the significant half of the dividend while the original contents of A is the insignificant half of the dividend. The divisor is the contents of the effective address. A 16-bit quotient will appear in the A Register while the final contents of the Q Register will contain the remainder. The sign of the quotient is determined in accordance with accepted division rules whereas the sign of the remainder will always be the same as the sign of the original dividend.

For the 1700, DVI instructions will produce a negative zero only under the following conditions:

(+0) ÷ (-N) = (-0), R = (+0)
(-0) ÷ (+N) = (-0), R = (-0)
(-2N) ÷ (+N) = (-2), R = (-0)

Overflow can occur during a DVI instruction. Consequently, overflow is monitored by the computer and the status recorded. Once overflow has been detected an overflow condition will continue to exist until either a SØV or SNØ instruction is executed.

Store Q (STQ) --

4	M	—Δ—

The Store Q (STQ) instruction provides a means of placing the contents of the Q Register into a memory location, the location being the effective address formed during the addressing mode of the instruction. If the location to which the contents of Q are to be stored is a protected location, the STQ instruction must be protected (that is, the protect switch OFF or the location of the STQ instruction be protected). If the STQ instruction is unprotected, an attempt to store its contents into a protected location will be illegal and will cause a protect fault. The effect of an illegal store is to convert the instruction to a pass by preventing any new transfer into memory. As a result, the original contents of the location referenced will remain unchanged.

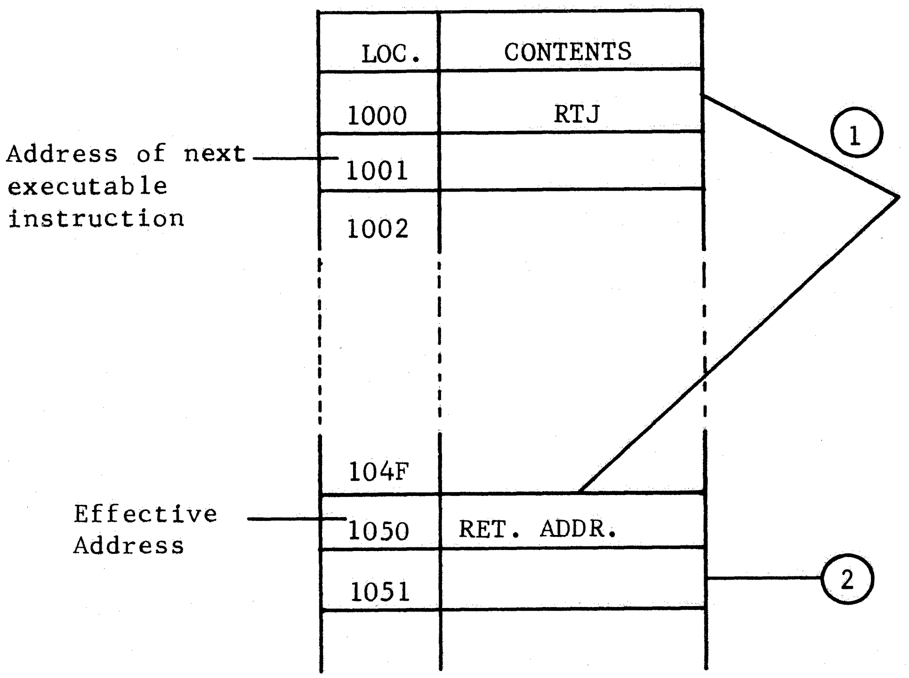
NOTE: An illegal store will not force the computer to execute a SLS instruction, as was true for other causes of protect faults.

Parity errors are also monitored during store instructions. Should a parity error exist and the STQ instruction is protected, the contents of Q will be stored but the parity error will persist until cleared by a skip instruction which senses parity status (SPE or SNP). Should a parity error exist but the STQ instruction is not protected, the operation is converted to a pass. The philosophy being that the parity error could have been caused by the protect bit changing status (from protected to non-protected) and a non-protected instruction must never be allowed to change the contents of a protected location.

Return Jump (RTJ) --

5	M	—Δ—

The Return Jump (RTJ) instruction is a branching instruction which provides for a return to the exact point where branching occurred. When the RTJ instruction is executed, the addressing mode will form an effective address which will specify the location for storing the return address. The return address will be the location of the next executable instruction (P+1 or P+2, depending on the address mode used to form the effective address). Program execution will continue at the location following the effective address. For example, assume a RTJ instruction is located at address 1000 and the effective address is 1050. If the next executable instruction is at location 1001, the following sequence would occur:



1 Addressing mode forms effective address (1050) and writes in that location the return address (the address of the next executable instruction -- 1001).

2 Program execution continues by executing the instruction at the effective address +1.

Once the reason for branching has been fulfilled, a return to the preceding program can be performed simply by executing a JMP instruction whose effective address is the return address written into location 1050 (for this example).

The RTJ instruction is a powerful instruction for the programmer in that it allows frequently used routines to be accessed from any location in memory and as many times as desirable without being concerned with the proper point of return.

The RTJ instruction is treated as a store instruction and must conform to protection requirements. If the location where the return address is to be stored is protected, the RTJ instruction must be protected or it becomes illegal, producing a protect fault and preventing the instruction from storing the return address. If a parity error exists when a non-protected RTJ instruction is attempted, the instruction is prevented from storing the return address. In both cases program continuation will be at the effective address +1. The effect, then, is that execution of RTJ instructions under fault conditions will prevent writing the return address into memory; however, branching will occur to the effective address +1.

Store A (STA) --

6	M	——Δ——

The Store A (STA) instruction provides a means of placing the contents of the A Register into the memory location specified by the effective address (formed during the addressing mode of the instruction). Execution of the STA instruction under fault conditions is the same as previously explained for the STQ instruction.

Store A, Parity to A (SPA) --

7	M	——Δ——

The SPA instruction provides a means of determining the parity of whatever is contained in the A Register. Execution of the SPA instruction will cause the contents of A to be stored at the location specified by the effective address. During this store operation, a parity bit will be generated for memory. This parity includes the memory protect bit at the location where (A) are stored. This is not necessarily the parity for the quantity originally in the A Register. To eliminate the effect of the protect bit, the parity bit and protect bit are exclusively ORed together. The result reflects the proper parity for the contents of A alone. The A Register is completely cleared, then the proper parity is transferred to A_{00}. Execution of the SPA instruction under fault conditions is the same as previously explained for the STQ instruction.

Add (ADD) --

8	M	——Δ——

The ADD instruction forms a 16-bit arithmetic sum of the contents of the A Register (Augend) and the contents of the memory location specified by the effective address (Addend). The sum replaces the original contents of the A Register.

For the 1700, the ADD instruction will produce a negative zero only under the condition:

$$(-0) + (-0)$$

Overflow is monitored during all ADD instructions and will occur when:

$$(-) + (-) = +$$
$$(+) + (+) = -$$

NOTE: If the address code (M) = 1, 2, or 3 and Δ = 0, the effective address so formed becomes the 16-bit quantity added to the contents of the A Register. This called the Immediate Operand mode and eliminates the need to reference memory for the Addend.

Subtract (SUB) --

9	M	——Δ——

The SUB instruction forms a 16-bit arithmetic difference between the contents of the A Register (Minuend) and the contents of the effective address

(Subtrahend). The difference replaces the original contents of the A Register.

For the 1700, the SUB instruction will produce a negative zero only under the condition:

(-0) - (+0)

Overflow is monitored during all SUB instructions and will occur when:

(-) - (+) = +
(+) - (-) = -

NOTE: If the address code (M) = 1, 2, or 3 and Δ = 0, the effective address so formed becomes the 16-bit quantity subtracted from the contents of the A Register.

And With A (AND) --

A	M	——Δ——

The AND instruction forms the logical product (AND) of the contents of the A Register and the contents of the effective address. The logical product replaces the original contents of the A Register.

NOTE: If the addressing code (M) = 1, 2, or 3 and Δ = 0, the effective address so formed becomes the 16-bit quantity ANDed with the contents of the A Register.

Exclusive OR With A (EØR) --

B	M	——Δ——

The EØR instruction forms the Exclusive OR of the contents of the A Register and the contents of the effective address. The exclusive OR result replaces the original contents of the A Register.

NOTE: If the addressing code (M) = 1, 2, or 3 and Δ = 0, the effective address so formed becomes the 16-bit quantity exclusively ORed with the contents of the A Register.

Load A (LDA) --

C	M	——Δ——

The LDA instruction provides a means of replacing the contents of the A Register with a quantity read from memory. The memory location referenced is the effective address.

NOTE: If the addressing code (M) = 1, 2, or 3 and Δ = 0, the effective address so formed becomes the 16-bit quantity which replaces the contents of the A Register.

Replace Add One (RAØ) --

D	M	——Δ——

The RAØ instruction reads the contents of the memory location specified by the effective address, adds +1 to the quantity and stores the incremented result at the location originally referenced. Execution of this instruction will always require two memory references following the addresssing mode. The first reference is to fetch the quantity to be incremented, the second reference is to store the incremented quantity.

The RAØ instruction is treated as a store instruction and must conform to protect requirements. If the instruction references a protected memory location, the instruction must be protected (the protect switch OFF or the location of the RAØ instruction be protected) to be executed. If a non-protected RAØ instruction references a protected location, the instruction is illegal. A protect fault is produced and the instruction becomes a pass. If a parity error exists during execution of a non-protected RAØ instruction, the instruction becomes a pass. However, even under fault conditions, the instruction will make two memory references. Overflow can exist as the result of an RØA instruction.

The RAØ instruction is useful in counting sequences where it becomes impractical to use the contents of a register for incrementing. When the count reaches a predetermined value, the sequence terminates.

Load Q (LDQ) -- $\boxed{\text{E} \mid \text{M} \mid \text{——}\Delta\text{——}}$

The LDQ instruction provides a means of replacing the contents of the Q Register with a quantity read from memory. The memory location referenced is the effective address.

NOTE: If the addressing code (M) = 1, 2, or 3 and Δ = 0, the effective address so formed becomes the 16-bit quantity which replaces the contents of the Q Register.

Add to Q (ADQ) -- $\boxed{\text{F} \mid \text{M} \mid \text{——}\Delta\text{——}}$

The ADQ instruction forms a 16-bit arithmetic sum of the contents of the memory location specified by the effective address (Augend) and the contents of the Q Register (Addend). The sum replaces the original contents of the Q Register.

For the 1700, the ADQ instruction will produce a negative zero only under the condition:

$$(-0) + (-0)$$

Overflow is monitored during all ADQ instructions and will occur when:

$$(-) + (-) = +$$
$$(+) + (+) = -$$

NOTE: If the addressing code (M) = 1, 2, or 3 and Δ = 0, the effective address so formed becomes the 16-bit quantity to which the contents of the Q Register are added.

CHAPTER III

ADDRESSING MODES

ADDRESSING MODES

INTRODUCTION

The execution of all addressable instructions requires the formation of an effective address. Formation of such an address uses the addressing mode capability of the 1700 computer. Since one hexadecimal digit (F) of each addressable instruction is used to identify the instruction, a maximum of 12 bits would be available as an address, which is 3 bits less than is necessary to reference 32K of memory. Because of the bit limitation in the instruction word size, various addressing modes have been incorporated to enable addressing capabilities to suit any need. The addressing mode is determined by another hexadecimal digit (M) which is a code to inform the computer the manner in which it is to arrive at the effective address. With two digits of the instructions being used for coding, only 8 bits are left for addressing. These 8 bits are referred to as the delta (Δ) field or modifier.

The format of all addressable instructions is:

F	M	—Δ—

where F must not equal 0 and M has the format:

11	10	9	8
r	ind	q	i

During each Read Next Instruction (RNI) mode, an instruction is read from memory, placed into the 1704 X Register after which the upper eight bits of the instruction transfers to the F Register. The contents of the F Register are immediately translated to determine which instruction is to be executed. If the instruction is addressable, the addressing mode is immediately started.

The addressing flow diagram (Appendix, page 14) shows the sequence during each addressing mode to arrive at the effective address. Each addressing code possibility is explained in the following pages.

RELATIVE

Relative, $\Delta \neq 0$

Consider the Load A (LDA) instruction C 8 3 5 . The function code (C) identifies the instruction as LDA while the addressing code in this example is the hexadecimal digit 8. In binary, this is equivalent to 1000. Since the only bit set is 2^{11}, it identifies the addressing mode as relative only. The lower two digits (35) represent the quantity Δ .

As soon as the computer determines that the instruction is addressable (the upper-most hexadecimal digit being non-zero), the addressing bits are considered--highest order bit first. If the relative bit is set, Δ is translated to determine whether or not it equals +0. In this example, Δ does not equal +0; consequently, its sign is extended (bit 2^7 determines the sign of Δ) to form a 16-bit quantity which is added to the contents of P.

The result of $P \pm \Delta$ is transferred to the Y register. If the only addressing bit set is the relative bit, then $P \pm \Delta$ (that is, the contents of Y) is the effective address.

Once the effective address is determined, memory is referenced at that address for execution of the instruction. Therefore, if the LDA instruction used in this example were located at address 0100 of memory, the effective address would be 0135. The contents of location 0135 would be referenced and loaded into the A Register.

What if the instruction had been $\boxed{C}\boxed{8}\boxed{8}\boxed{3}$? In this case, the quantity Δ is negative (bit 2^7 is set) so that P (0100) plus Δ with sign extension (FF83) would become 0084.

Relative, $\Delta = 0$

What if the quantity Δ equals +0? Consider the LDA instruction $\boxed{C}\boxed{8}\boxed{0}\boxed{0}$ If the quantity Δ equals +0, relative addressing is performed in two steps. First, P + 1 is formed and sent to P and Y. This location (Y) is then referenced and its contents added to the present quantity in Y. Thus, if the instruction C800 is located at address 0100, +1 is first added to P and sent to P and Y. The contents of Y (location 0101) is referenced. The quantity read from memory is then added to Y, the relative bit is cleared and relative addressing ceases. In the example being used, if the contents of location 0101 were 0050, the result of relative addressing would produce:

$$P + 1 + (P + 1) = 0101 + 0050 = 0151.$$

$\overline{\text{Relative}}$, $\Delta = 0$

In the LDA instruction $\boxed{C}\boxed{0}\boxed{0}\boxed{0}$, the relative bit is clear and Δ equals zero. Under such a condition relative addressing forms P + 1, placing the result in P and Y. If the instruction C000 were at location 02AF, the result of relative addressing would produce 02B0 in the P and Y registers.

$\overline{\text{Relative}}$, $\Delta \neq 0$

In the LDA instruction $\boxed{C}\boxed{0}\boxed{B}\boxed{E}$, the relative bit is clear but Δ equals BE. Under such a condition relative addressing merely involves the transfer of $+\Delta$ to Y. In this example, then, the result of relative addressing would place the quantity 00BE in the Y Register.

1. In the following partial programs, what would be loaded into the A Register as a result of each LDA instruction?

```
a.  0222 = C8A4                     b.  0222 = C800
    0223 = 8800                         0223 = 00A4
    01C7 = 4321                         0044 = 4321
    1A23 = 1234                         02C7 = FFFF

c.  1234 = C801                     d.  1234 = C885
    1235 = 1500                         1235 = 1500
    1236 = 2000                         11BA = FF00
    1500 = FF00                         1500 = 2468
```

2. Explain the difference, if any, between the results obtained by the LDA instructions in each of the following partial programs.

```
a.  0500 = C800                     b.  0500 = C801
    0501 = 2233                         0501 = FFFF

    2233 = 00FF                         2233 = 00FF
    2734 = FFFF                         2734 = 2233
```

3. Where would each program (problem 2) continue after execution of the LDA instruction?

INDIRECT

Following each relative operation, the computer checks the status of the indirect bit. If this bit is set, it informs the computer that indirect addressing will commence where relative ends. If the indirect bit is not set, indirect addressing will not be performed.

The following examples show combinations of addressing modes with the indirect bit set. The first example shows non-relative, indirect and $\Delta = 0$.

```
0035 - F0F0
0100 = C400
0101 = 8102
0102 = 0035
```

The LDA instruction at location 0100 has a 4 addressing code, which means only the "ind" bit is set. Since r = 0 and $\Delta = 0$, relative operations would form P + 1 in P and Y. This ends relative operations, at which time indirect operations begin. Indirect addressing takes the contents of Y (0101 in this case), references memory and makes a sign check of the quantity when it gets into the X Register. If the sign bit (2^{15}) of X is set, during indirect operations, it means the quantity just read from memory is another indirect address. In this example the contents of location 0101 are negative. As a result, the contents of X (8102) are transferred to Y and used as another indirect address. Since only the lower 15 bits of Y have any significance regarding memory addressing, the actual location indirectly addressed this time will be 0102. This multi-level indirect operation continues until a positive quantity is read from memory. For this example, the contents of location 0102 are positive. Once a positive quantity is detected in X, the indirect bit is cleared causing further indirect operations to cease. The last quantity read from memory transfers to Y and awaits further addressing (indexing) if any.

In the preceding example, the C400 instruction would result in an effective address (in Y) of 0035. Since the instruction is a LDA, location 0035 would be referenced and its contents (F0F0) placed into the A Register.

The next example shows non-relative, indirect and $\Delta \neq 0$.

```
0002 = 0101
0035 = 0102
0100 = C402
0101 = 8102
0102 = 0035
70F0 = 0100
```

In this instance the result of relative operations places $+\Delta$ into Y; that is, Y would equal 0002. Indirect operations use the contents of Y as an indirect address which is referenced in the manner previously explained. The result of the C402 instruction would be that the Effective Address would be 0101. The contents of this location (8102) would be loaded into the A Register.

The following partial program shows a LDA instruction which has relative, indirect and $\Delta = 0$.

```
0100 = CC00
0101 = 7FFF
0102 = 0123
4C00 = 0101
```

The CC00 instruction will form P + 1 in P and Y. The contents of Y are referenced obtaining the quantity 7FFF which is added to Y. This makes Y equal 8100, which is the indirect address. When memory is referenced, the upper-most bit in Y is discarded so that (for this example) location 0100 will be referenced. Since the quantity CC00 is negative, indirect addressing is repeated at location 4C00. This produces an effective address of 0101 so that the quantity 7FFF is loaded into the A Register.

The following indirect address example involves a LDA instruction having relative, indirect and $\Delta \neq 0$.

```
00AB = 8103
00AC = 8102
0100 = CCAB
0101 = 00AB
0102 = 00AC
0103 = 8101
```

The CCAB instruction will form $P \pm \Delta$ in Y, making Y equal 00AC. This location is indirectly addressed. Since its contents (8102) is negative, indirect addressing is repeated at location 0102. The effective address becomes 00AC so that the quantity 8102 is loaded into the A Register.

- - - - - - - - - - - -

1. Repetitive (multi-level) indirect addressing is indicated in what manner?

2. What would be placed into the A Register as a result of the LDA instruction in the following partial program?

```
0000 = 0088
0088 = C489
0089 = 8090
0090 = 9000
1000 = 9001
1001 = FFFF
7FFF = 1234
```

3. In the preceding program, what would happen if the contents of location 1001 were 9000?

4. See if you can determine the quantity loaded into the A Register due to the LDA instruction in each of the following partial programs.

```
a.  00FF = 8101                  b.  0000 = 8102
    0100 = CCFF                      0100 = C400
    0101 = 8805                      0101 = 4321
    0102 = 1234                      0102 = 0101
    01FF = 80FF                      4321 = 0F0F
    4805 = 00FF
    4CFF = 0100

c.  0000 = CC00                  d.  0000 = CC01
    0001 = 8002                      0001 = 8000
    0002 = FFFF                      0002 = 1234
    0003 = FC00                      4C01 = 0000
    7C00 = 0001                      8000 = 0001
    7FFF = 0002
```

INDEXING

Two index registers are available for addressing purposes in the 1700 Computer. One is the Q Register and is selected by having the "q" bit (2^9) of the addressing code set. The second register is the contents of memory location 00FF and is selected by having the "i" bit (2^8) of the addressing code set. If both bits are set, the contents of both registers will be used with (Q) being used first.

Consider the LDA instruction $\boxed{C}\boxed{2}\boxed{8}\boxed{0}$. Since the addressing code is 2, only the "q" bit is set. Δ = 80 which becomes 0080 with zeros extended. With only the Q bit set in the addressing code, the effective address is formed by adding the contents of Q to $+\Delta$. Thus, the effective address in this particular example (assuming Q = 1234) would be 0080 + 1234 or 12B4.

> NOTE: The largest possible address in the computer is 7FFF (15 bits). Consequently, whenever the output of the adder is an address, it will be reduced to 15 bits by dropping the most significant (sign) bit. This occurs during the Y to S transfer only.
>
> Example: $\boxed{C}\boxed{2}\boxed{8}\boxed{0}$, Q = $\boxed{C}\boxed{2}\boxed{8}\boxed{0}$
>
> The result of 0080 + C280 would be C300. However, the memory location referenced would be 4300.

Let's consider the following partial program to illustrate indexing.

```
   Q = 0101
0100 = C201
0101 = 1234
0102 = FF00
```

In the preceding program the LDA instruction uses Q indexing and Δ = 01. Q + Δ = 0102 which is the effective address. The contents of 0102 are then loaded into the A Register.

The LDA instruction $\boxed{C}\boxed{1}\boxed{5}\boxed{0}$ has an addressing code of 1, which identifies the use of the memory index register (location 00FF). If the contents of 00FF = 8500, the effective address would be formed by adding 8500 to 0050, giving 8550. This, in turn, is reduced to 15 bits (Y to S) giving an effective address of 0550. The contents of 0550 would then be loaded into the A Register.

The LDA instruction $\boxed{C}\boxed{3}\boxed{7}\boxed{A}$ has an addressing code of 3, which identifies the use of both indexing registers. Assuming the contents of Q to be CA7D and the contents of location 00FF to be DEFF, the effective address would be formed in the following manner:

$$\begin{array}{rrl} \triangle \text{ with zeros extended} = & \text{007A} & \\ \text{Contents of Q} = & \underline{\text{CA7D}} & \\ \text{16-bit sum} = & \text{CAF7} & \\ & & \\ \text{Contents of 00FF} = & \underline{\text{DEFF}} & \\ & \text{A9F6} & \\ & \underline{\quad\quad 1} & \text{-- end around carry} \\ \text{Effective Address} \quad = & \text{A9F7} & \text{which, when} \\ & & \text{reduced to 15 bits} \\ & & \text{becomes 29F7.} \end{array}$$

The contents of location 29F7 would then be loaded into the A Register.

What would be the contents of the A Register due to the LDA instruction in each of the following partial programs?

1. Q = 89FA
00FF = 776F
0100 = C399
0101 = 1234
0203 = FFFF

2. Q = FEDB
00FF = 0057
0100 = C277
0101 = 1234
7F52 = 2468

3. Q = 0000
00FF = 0102
0100 = C100
0101 = 1234
0102 = DDFF
0203 = ABCD

4. Q = 00FF
00FF = FF01
0100 = C3FF
0101 = 1234

MULTIPLE ADDRESSING

Each addressing mode considered previously has been designated by individual bits of the addressing code. We have seen that relative and indirect addressing modes can be combined. We have also seen that both index registers may be used in one operation. Can indexing be combined with relative? With indirect? With both? Yes, they can be combined. It is this multiple addressing capability which provides a powerful tool to the 1700 programmer. The basic rule to remember in multiple addressing is that each mode is performed using the results obtained in the previous mode. The order of operation is:

1. Relative
2. Indirect
3. Q indexing
4. Memory indexing

Let's consider the following partial program:

```
   Q = 0302
OOFF = 0100
0500 = CF00
0501 = 8503
0502 = 00FF
0503 = FFFF
0A04 = 80FF
```

In the preceding program, the CF00 instruction has all addressing bits set. The manner in which the effective address is formed is as follows:

1. Form P + 1 in P and Y since relative set and $\Delta = 0$
2. Reference location 0501.
3. Add contents of location 0501 to contents of Y, giving Y = 8A04
4. Indirect at location 0A04 (contents of Y reduced to 15 bits)
5. Indirect at location 00FF (since the contents of 0A04 are negative).
6. Stop indirect operations (since contents of 00FF are positive) and sum the contents of 00FF with Q, giving 0402.
7. Sum 0402 with the contents of memory index register 00FF, giving 0502.
8. Load the contents of 0502 into the A Register, making A = 00FF.

Another example of multiple addressing is given in the following partial program.

```
   Q = 33FD
00FF = 2510
1000 = C300
1001 = 8105
1002 = 1234
5A12 = FF44
```

If you have tried the preceding partial program, you probably concluded that the quantity FF44 is loaded into the A Register. However, this is not true. The actual quantity loaded into A is the final contents of Y, DA12. For a moment, look back at the addressing flow diagram (Appendix) and re-read the note. In effect, then, whenever relative with △ = 0 and indexing but not indirect, THE EFFECTIVE ADDRESS BECOMES AN OPERAND IF THE INSTRUCTION NORMALLY READS AN OPERAND FROM MEMORY. Bear in mind that it is the contents of P + 1 to which (Q) and/or (00FF) are added. Remember, also, the effective address is the contents of the Y register upon completion of all addressing operations. The manner in which the preceding program would execute the LDA instruction is as follows:

1. Relative and △ = 00, form relative address P + 1 (1001).
2. Add the contents of Q to the quantity read from location 1001; that is, 8105 + 33FD, giving B502.
3. Add the contents of memory index register (location 00FF) to B502; that is, B502 + 2510, forming DA12 in the Y Register. Since the instruction is one that normally reads an operand from memory, the effective address (final contents of Y) becomes the quantity which is loaded into the respective register. Consequently, the A Register will be loaded with the quantity DA12. Notice that Immediate Operand operations produce a 16-bit quantity.

Determine the quantity placed in the A Register during execution of the LDA instruction in each of the following partial programs.

```
1.     Q = 57AF          2.     Q = 7FFE
    00FF = FFFE              00FF = 543E
    0100 = C500              1000 = C200
    0101 = 8102              1001 = 9002
    0102 = 0101              1002 = 0100

3.     Q = 57AF          4.     Q = 87FE
    00FF = 0500              00FF = F80E
    0500 = C101              0750 = CBF2
    0501 = 9002              0751 = 0246
    0502 = 0500              0752 = 9999

5.     Q = 1253          6.     Q = 1F02
    00FF = 0001              00FF = 1F01
    1250 = 1251              2000 = A002
    1251 = FFFF              2001 = CEFE
    1252 = C9FE              2002 = 00FF
    1253 = 1250              2003 = FFFF
```

1. What is the final contents of the A, Q and P Registers for each of the following routines?

   a. Initial Conditions: M.C., Set P = 0500, Selective Stop switch ON and press RUN.

```
04FF --- 0000
0500 --- 0A89
0501 --- 0C01
0502 --- 08B1
0503 --- 0FE8
0504 --- 08B2
0505 --- 0F4B
0506 --- 0881
0507 --- 9200
0508 --- 0500
0509 --- 0131
050A --- 0000
050B --- 1400
050C --- 04FF
```

   b. Initial Conditions: M.C., Set P =0200, Selective Stop switch ON, Skip Switch ON, Set A = 0003, press RUN

```
01FF --- 0000
0200 --- 0C7F
0201 --- 0D7F
0202 --- 09FE
0203 --- 0101
0204 --- 18FC
0205 --- 0D04
0206 --- 88F9
0207 --- 08F1
0208 --- 0838
0209 --- 01B1
020A --- 0EFF
020B --- B000
020C --- 0202
020D --- 0000
```

   c. Initial Conditions: M.C., Set P = 1500, Selective Stop switch ON and press RUN

```
00FF --- 0002
1500 --- 0AF0
1501 --- 0D79
1502 --- 08F9
1503 --- E100
1504 --- 1500
1505 --- 9000
1506 --- 00FF
1507 --- 0879
1508 --- F400
1509 --- 9506
150A --- 78E0
150B --- 0000
```

d. Initial Conditions: M.C., Set P = 2000, Selective Stop switch ON, Protect switch ON, set A = 7085, Set Q = 5522, Set Mask = 0005 and press RUN

```
2000 --- 0807
2001 --- 0121
2002 --- 5000
2003 --- 2004
2004 --- 0000
2005 --- 0BFE
2006 --- 0000
```

e. Initial Conditions: M.C. Set P = 1700, Selective Stop switch ON and press RUN

```
1700 --- 0BF7
1701 --- 5806
1702 --- D8FE
1703 --- 0132
1704 --- 09FE
1705 --- 1400
1706 --- 1701
1707 --- 0000
1708 --- 18F9
1709 --- 0000
```

2. Write routines that will simulate the following 1700 instructions. You are free to use any 1700 instruction in each routine except the one you are simulating.

   a. And with A
   b. Load Q
   c. Subtract
   d. Exit the Interrupt State
   e. Long Left Shift
   f. Store A, send parity to A_{00}

ANSWERS -- CHAPTER III

Page 3-3

1. a. A = 4321
   b. A = FFFF
   c. A = 1500
   d. A = FF00

2. No differences. In either case A = FFFF.

3. Program "a" would continue at location 0502 while program "b" would continue at location 0501.

Page 3-5

1. Multi-level indirect addressing is indicated if the quantity indirectly addressed is negative. The sign bit of quantities indirectly addressed is a flag notifying the computer whether indirecting is to continue or to terminate.

2. The contents of A would equal whatever was stored at location 1234. Remember that the highest possible address with 32K of memory is 7FFF.

3. The computer would be caught in an endless indirect addressing loop between locations 1000 and 1001. Embarrassing, to say the least, for the programmer.

4. a. A = CCFF (Don't forget that the sign of Δ is extended and added to P if "r" and $\Delta \neq 0$ exist).

   b. A = 0F0F ($\bar{r}$ and $\Delta = 0$ forms P + 1 which is indirectly addressed.)

   c. A = 8002 (Don't forget relative and $\Delta = 0$)

   d. A = CC01 (Remember, there are no addresses greater than 7FFF in the 1700 system.)

Page 3-7

1. A = FFFF

2. A = 2468

3. A = 1336. This is a tricky one. In the C100 instruction, the condition $(r)(ind)(\Delta = 0)(q + i)$ is satisfied so that the effective address (formed by adding contents of P + 1 and contents of 00FF) becomes the quantity loaded into A.

4. A = C3FF

Page 3-9

1. A = C500

2. A = 1001

3. A = 9002 (Don't forget, Δ is not equal to zero.)

4. A = CBF2

5. A = C9FE

6. A = CEFE

Pages 3-12, 3-13, 3-14

1. a. (A) = F9F1 (Q) = 0100 (P) = 0500
   b. (A) = 0E7D (Q) = 0200 (P) = 020E
   c. (A) = 0001 (Q) = 1504 (P) = 150C
   d. (A) = 7085 (Q) = 5522 (P) = 2001

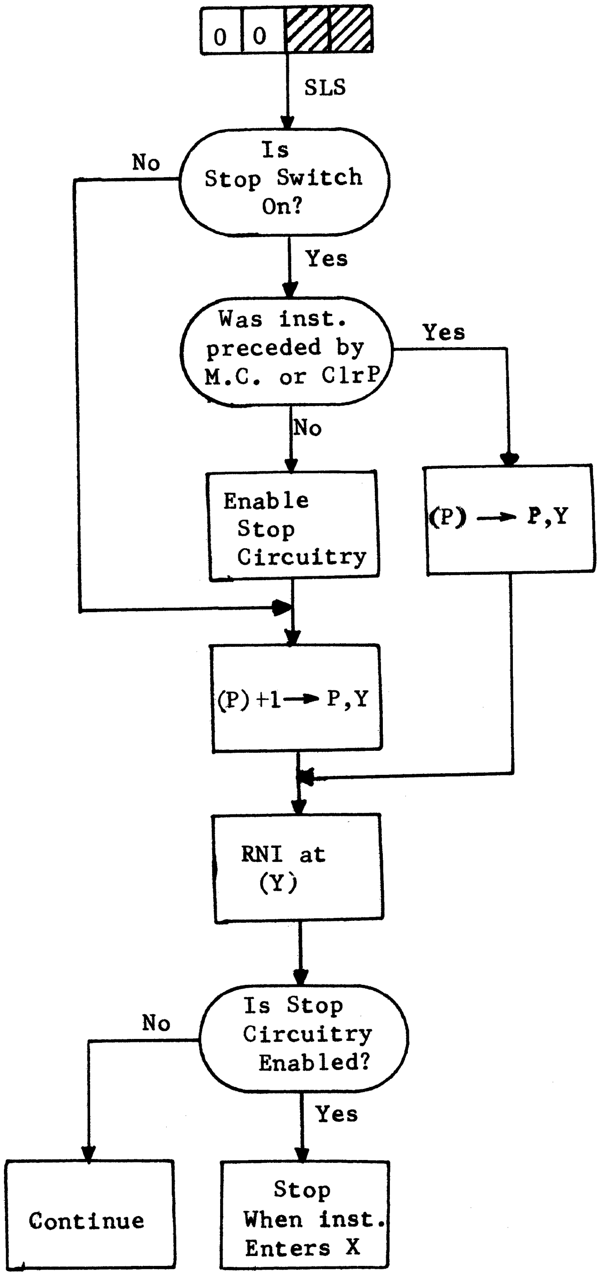
   NOTE: Instruction at location 2000 is illegal, causing computer to stop.

   e. (A) = FFFE (Q) = 0000 (P) = 170A

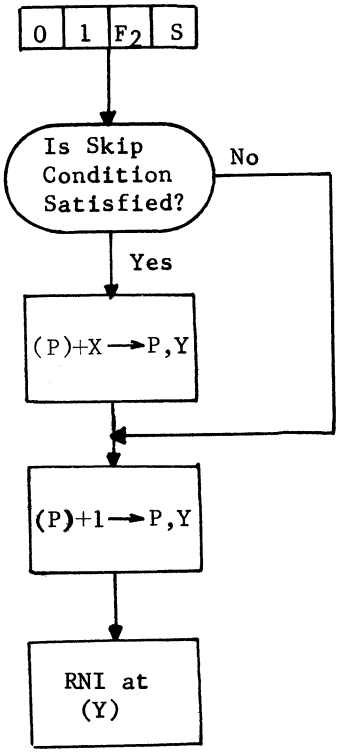
2. Routines will be checked by running them in the computer.

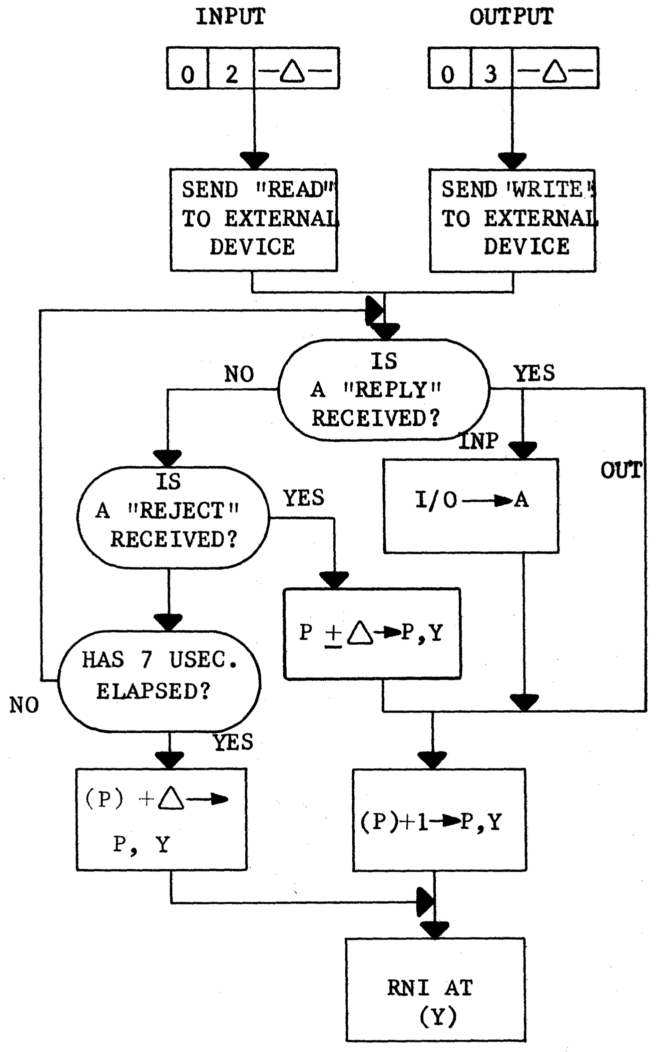
APPENDIX

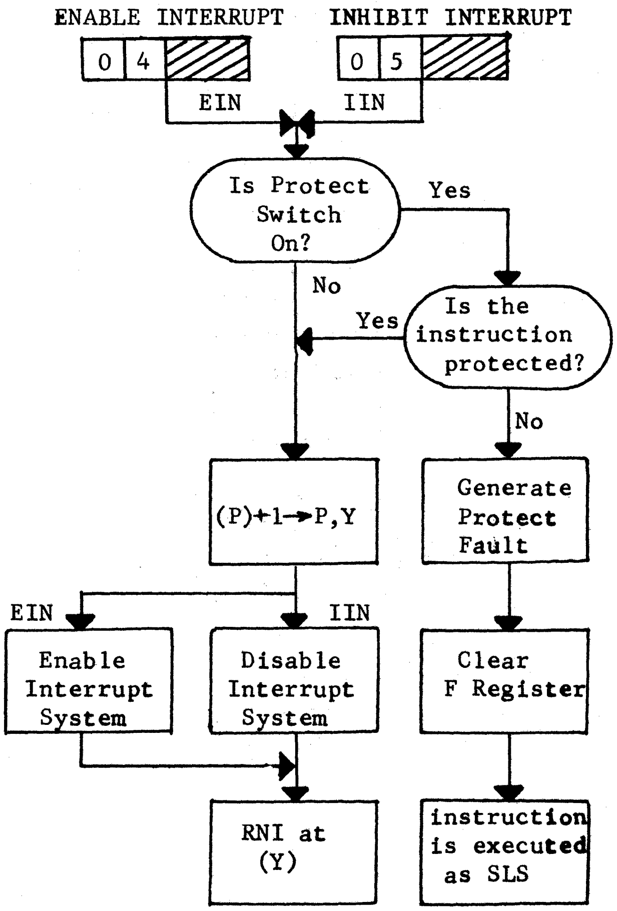
SELECTIVE STOP

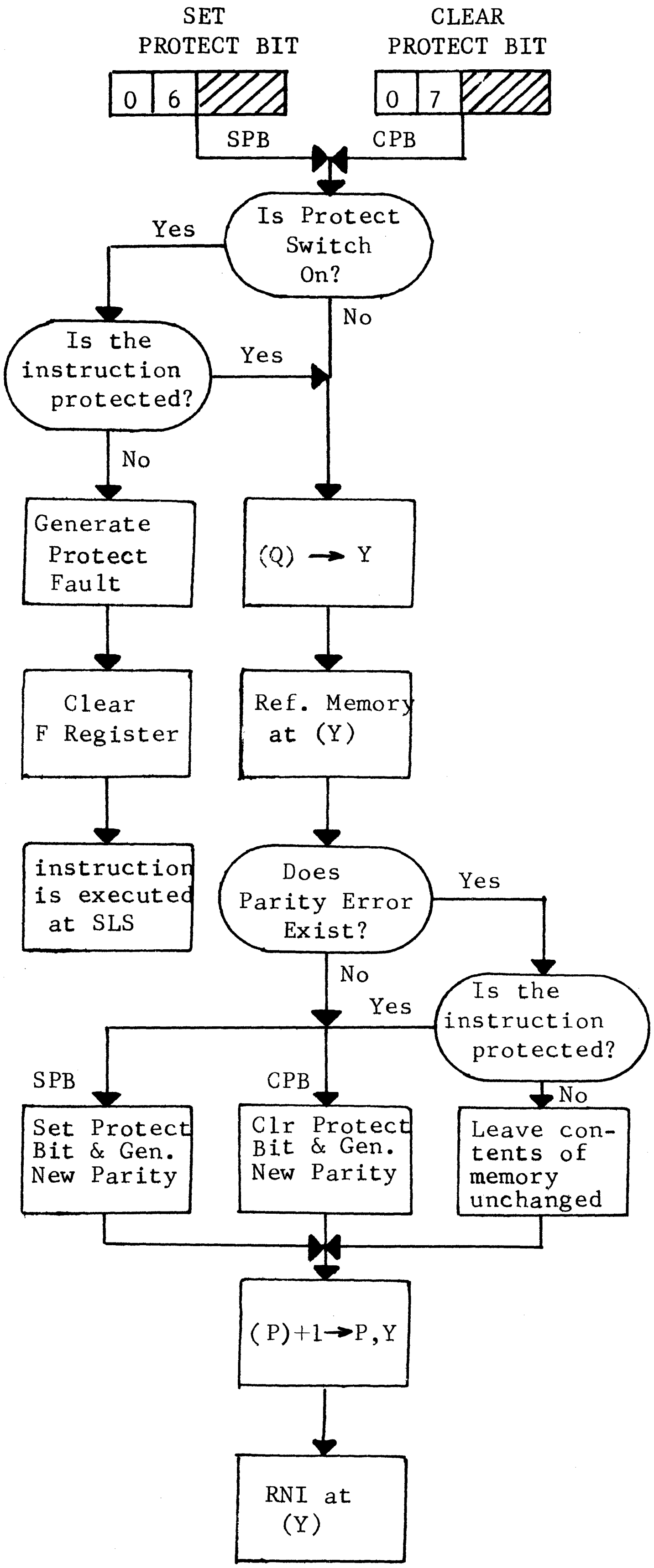


SKIP



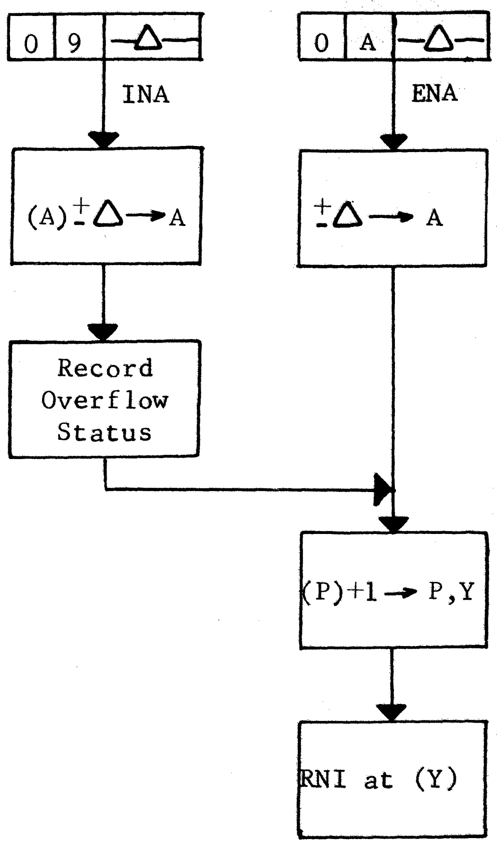

INPUT
OUTPUT
0 2 —△—
0 3 —△—
SEND "READ" TO EXTERNAL DEVICE
SEND 'WRITE' TO EXTERNAL DEVICE
IS A "REPLY" RECEIVED?
NO
YES
INP
OUT
I/O ⟶ A
IS A "REJECT" RECEIVED?
YES
P ±△ ⟶ P,Y
HAS 7 USEC. ELAPSED?
NO
YES
(P) +△ ⟶ P, Y
(P)+1 ⟶ P,Y
RNI AT (Y)


ENABLE INTERRUPT
INHIBIT INTERRUPT
0 4
0 5
EIN
IIN
Is Protect Switch On?
Yes
No
Is the instruction protected?
Yes
No
(P)+1 ⟶ P,Y
Generate Protect Fault
EIN
IIN
Enable Interrupt System
Disable Interrupt System
Clear F Register
RNI at (Y)
instruction is executed as SLS

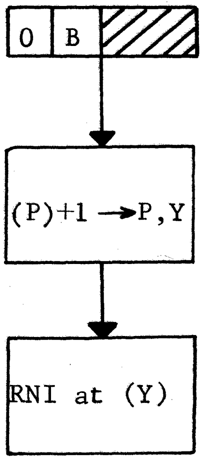

SET
PROTECT BIT
CLEAR
PROTECT BIT
0
6
0
7
SPB
CPB
Is Protect Switch On?
Yes
No
Is the instruction protected?
Yes
No
Generate Protect Fault
(Q) → Y
Clear F Register
Ref. Memory at (Y)
instruction is executed at SLS
Does Parity Error Exist?
Yes
No
Is the instruction protected?
Yes
No
SPB
CPB
Set Protect Bit & Gen. New Parity
Clr Protect Bit & Gen. New Parity
Leave contents of memory unchanged
(P)+1→P,Y
RNI at (Y)

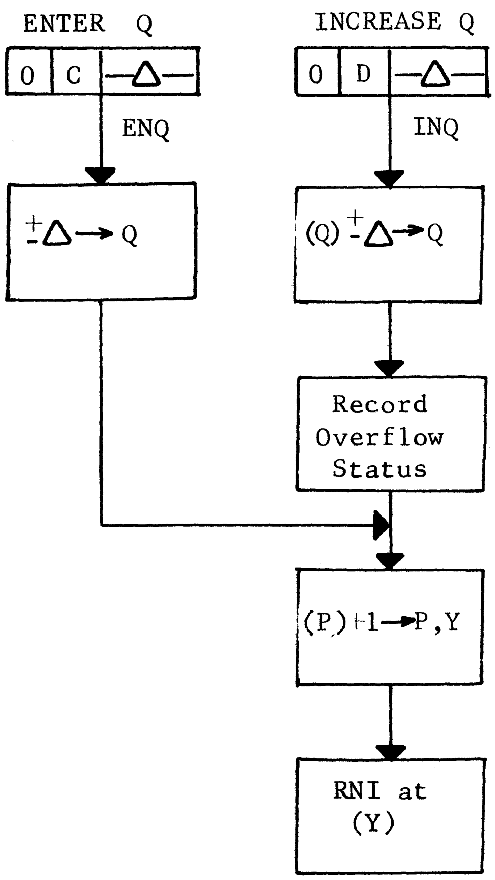
INCREASE A

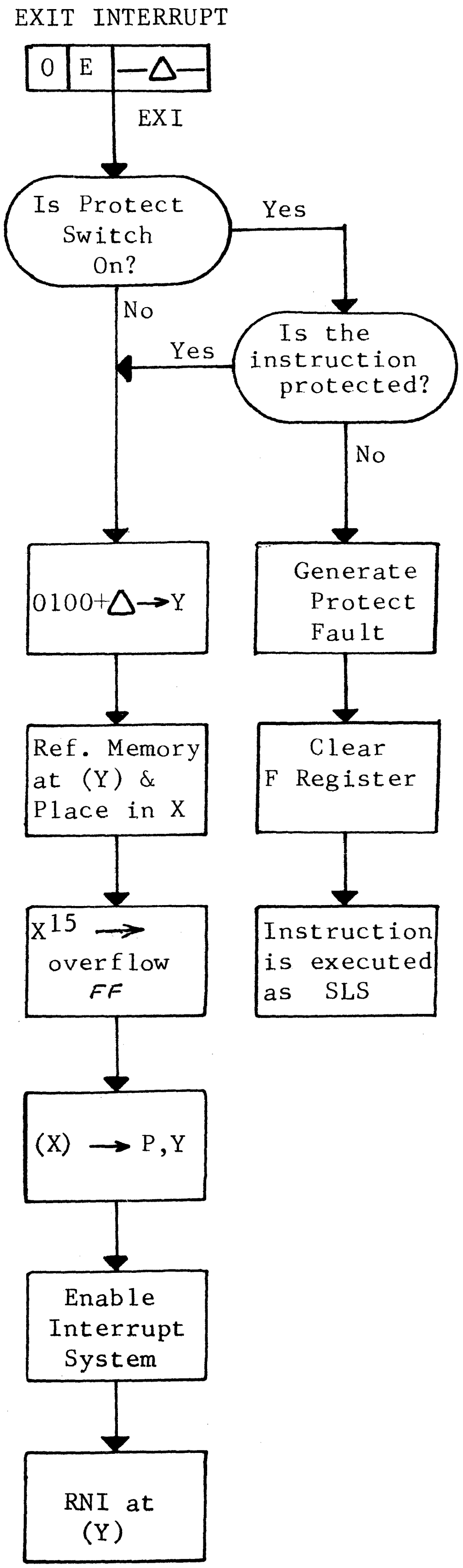
ENTER A



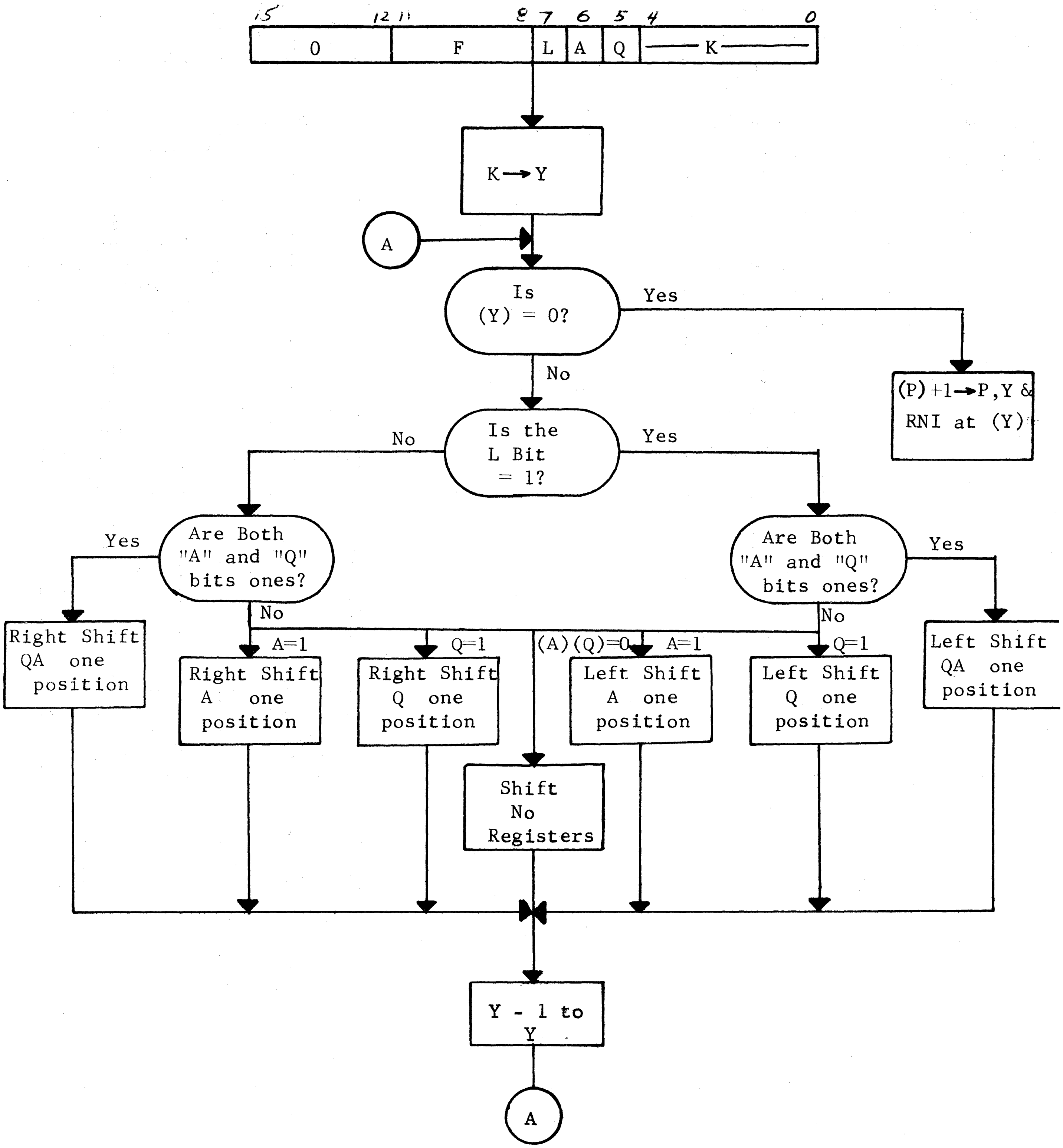
NO OPERATION



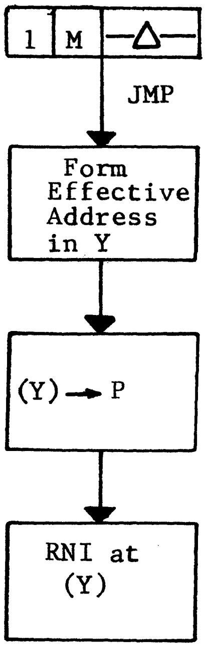

ENTER Q
0 | C | —Δ—
ENQ
±Δ → Q
INCREASE Q
0 | D | —Δ—
INQ
(Q) ±Δ → Q
Record Overflow Status
(P)+1 → P,Y
RNI at (Y)


EXIT INTERRUPT
0 | E | —Δ—
EXI
Is Protect Switch On?
Yes
No
Is the instruction protected?
Yes
No
0100+Δ → Y
Ref. Memory at (Y) & Place in X
X15 → overflow FF
(X) → P,Y
Enable Interrupt System
RNI at (Y)
Generate Protect Fault
Clear F Register
Instruction is executed as SLS

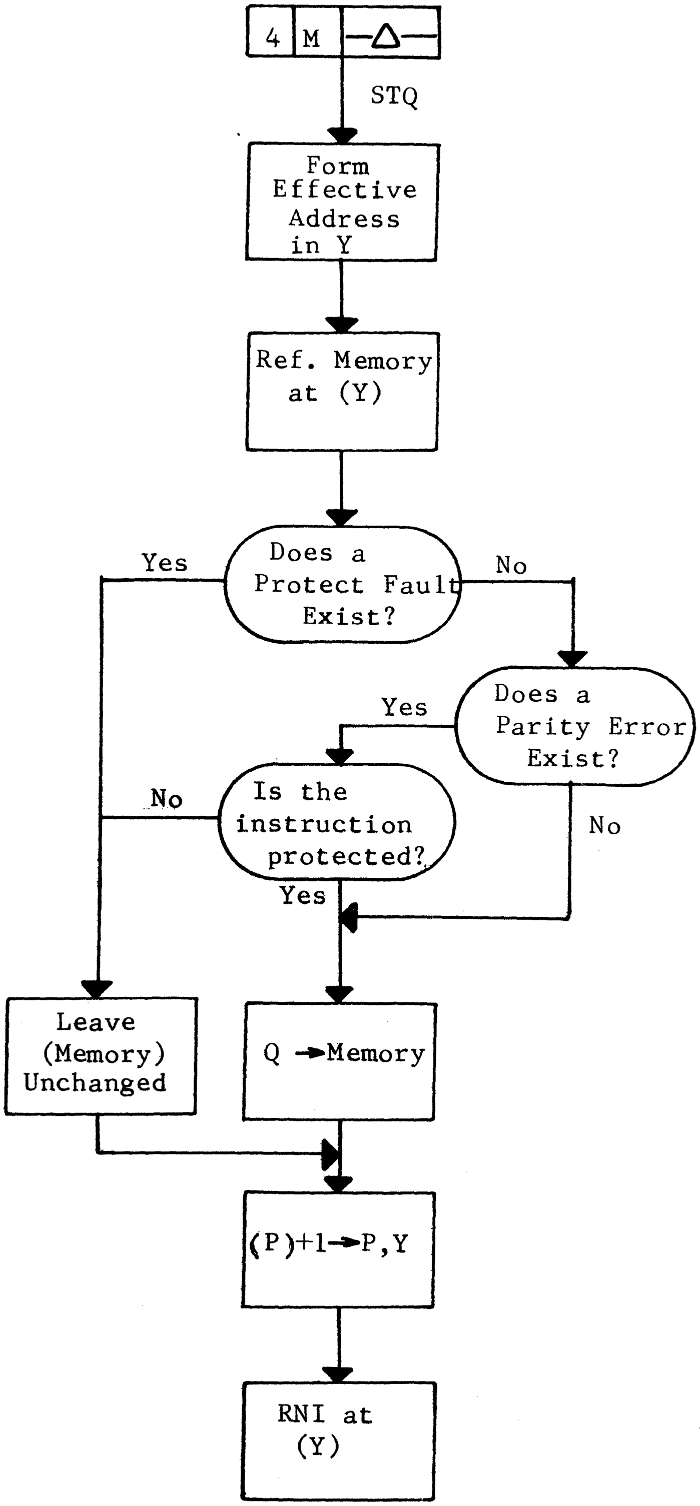
SHIFT

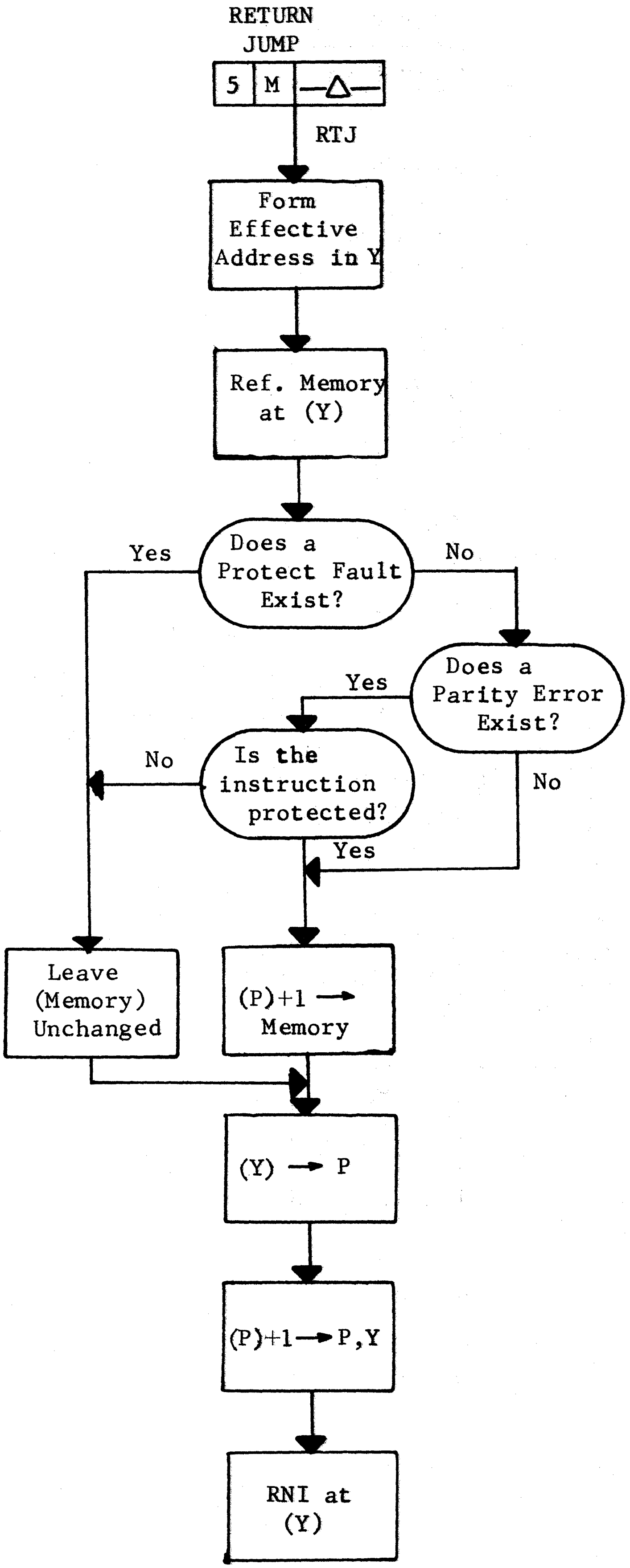

15
12 11
8 7
6
5
4
0
0
F
L
A
Q
K
K → Y
A
Is (Y) = 0?
Yes
(P) +1 → P, Y &
RNI at (Y)
No
Is the L Bit = 1?
No
Yes
Are Both "A" and "Q" bits ones?
Yes
Right Shift QA one position
No
A=1
Right Shift A one position
Q=1
Right Shift Q one position
(A)(Q)=0
Shift No Registers
Are Both "A" and "Q" bits ones?
Yes
Left Shift QA one position
No
A=1
Left Shift A one position
Q=1
Left Shift Q one position
Y - 1 to Y
A

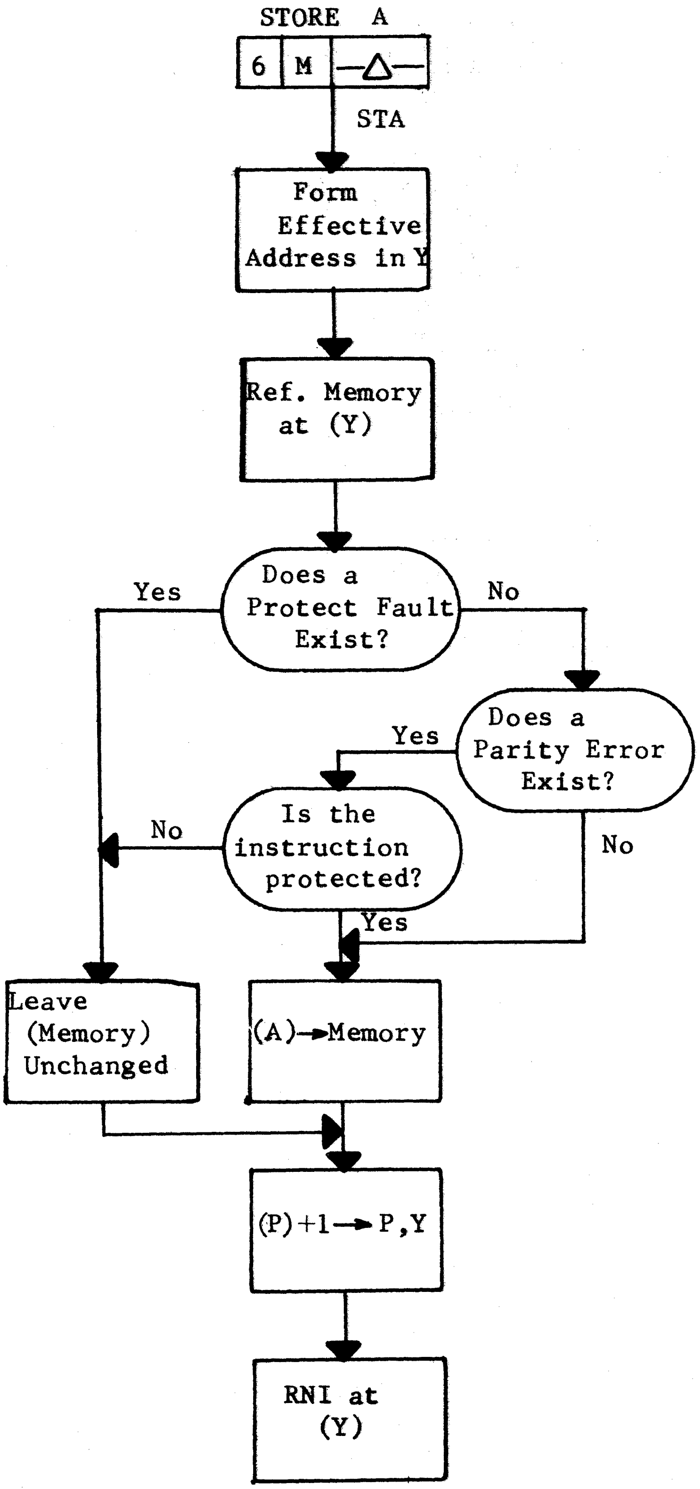
JUMP

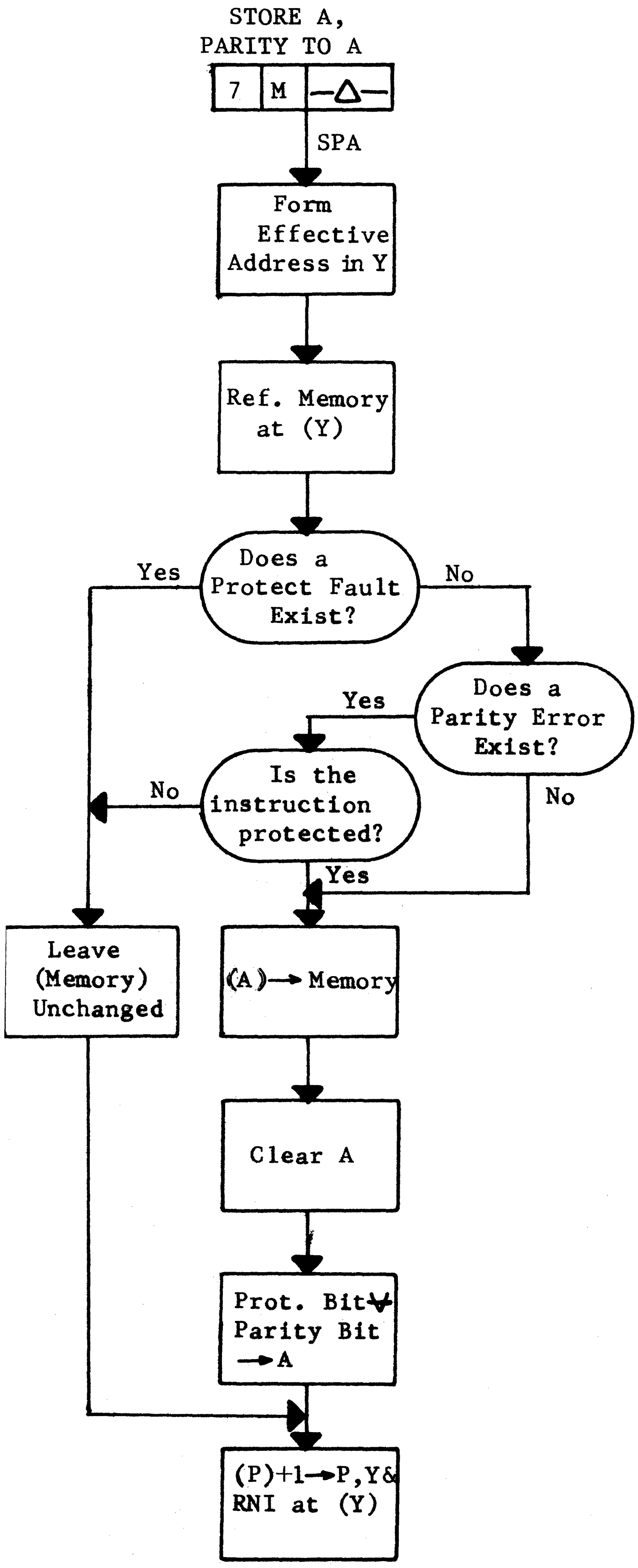

1
M
—Δ—
JMP
Form Effective Address in Y
(Y) → P
RNI at (Y)

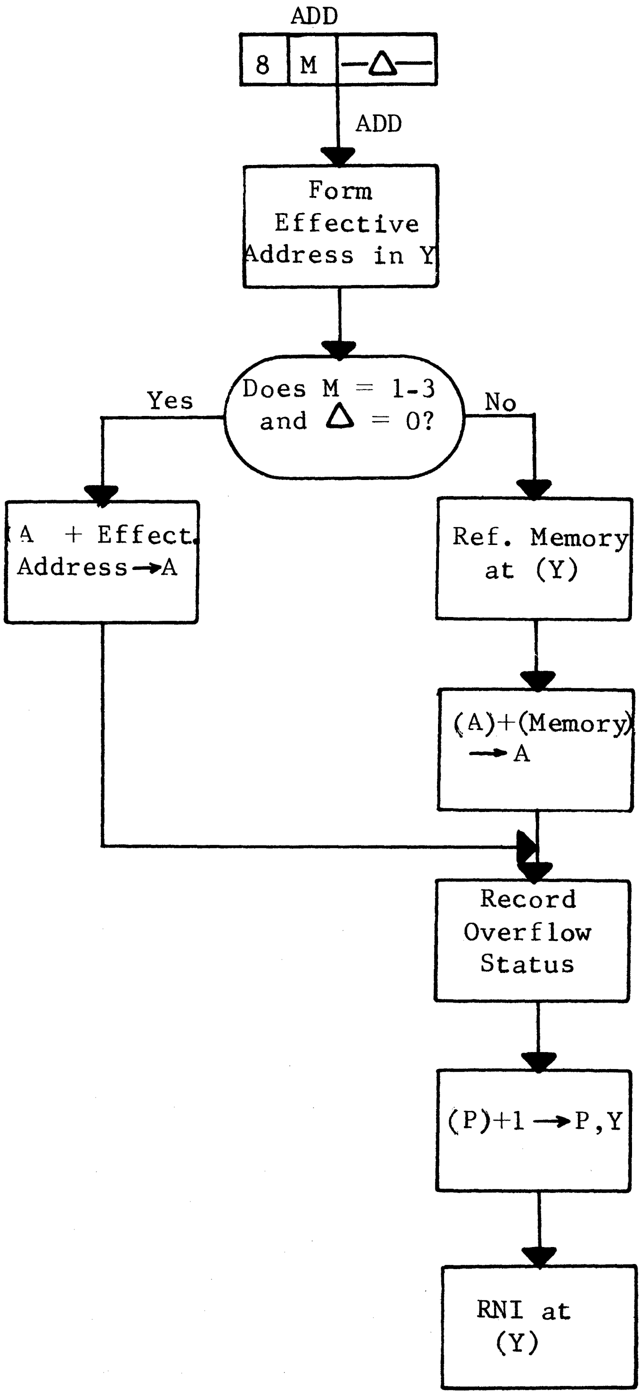
STORE Q


4
M
—Δ—
STQ
Form Effective Address in Y
Ref. Memory at (Y)
Yes
Does a Protect Fault Exist?
No
Yes
Does a Parity Error Exist?
No
Is the instruction protected?
No
Yes
Leave (Memory) Unchanged
Q → Memory
(P)+1 → P,Y
RNI at (Y)

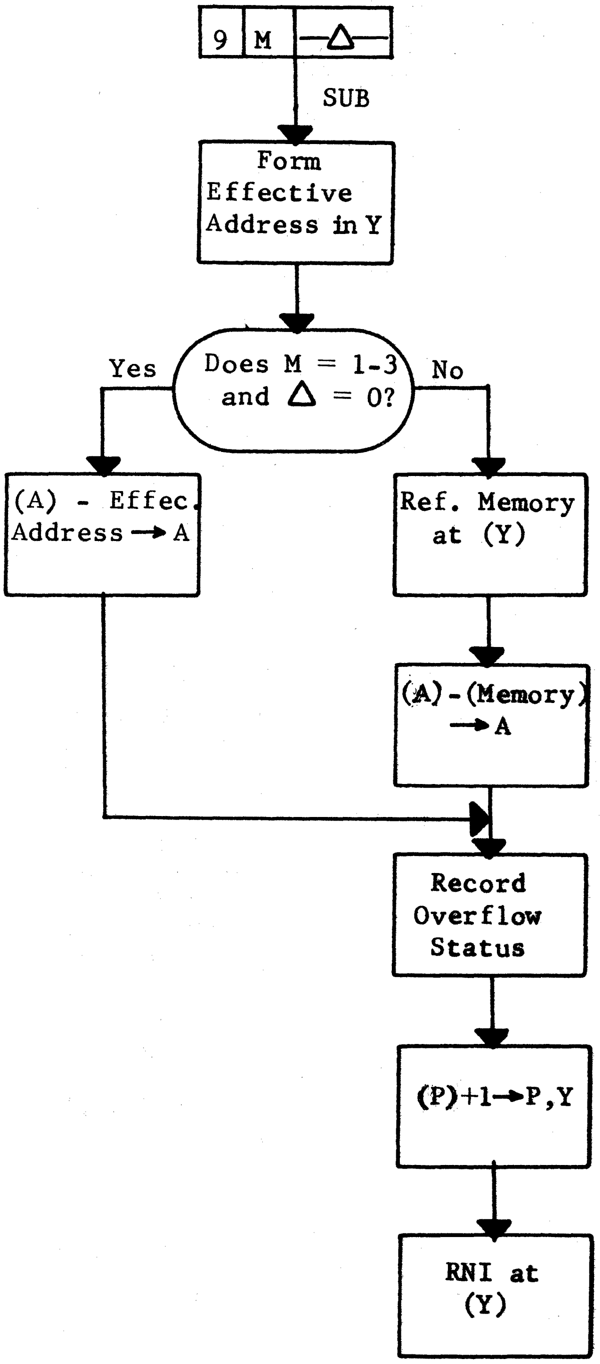

RETURN JUMP
5
M
—Δ—
RTJ
Form Effective Address in Y
Ref. Memory at (Y)
Does a Protect Fault Exist?
Yes
No
Does a Parity Error Exist?
Yes
No
Is the instruction protected?
No
Yes
Leave (Memory) Unchanged
(P)+1 → Memory
(Y) → P
(P)+1 → P,Y
RNI at (Y)


STORE A
6
M
—Δ—
STA
Form Effective Address in Y
Ref. Memory at (Y)
Does a Protect Fault Exist?
Yes
No
Does a Parity Error Exist?
Yes
No
Is the instruction protected?
No
Yes
Leave (Memory) Unchanged
(A)→Memory
(P)+1→P,Y
RNI at (Y)

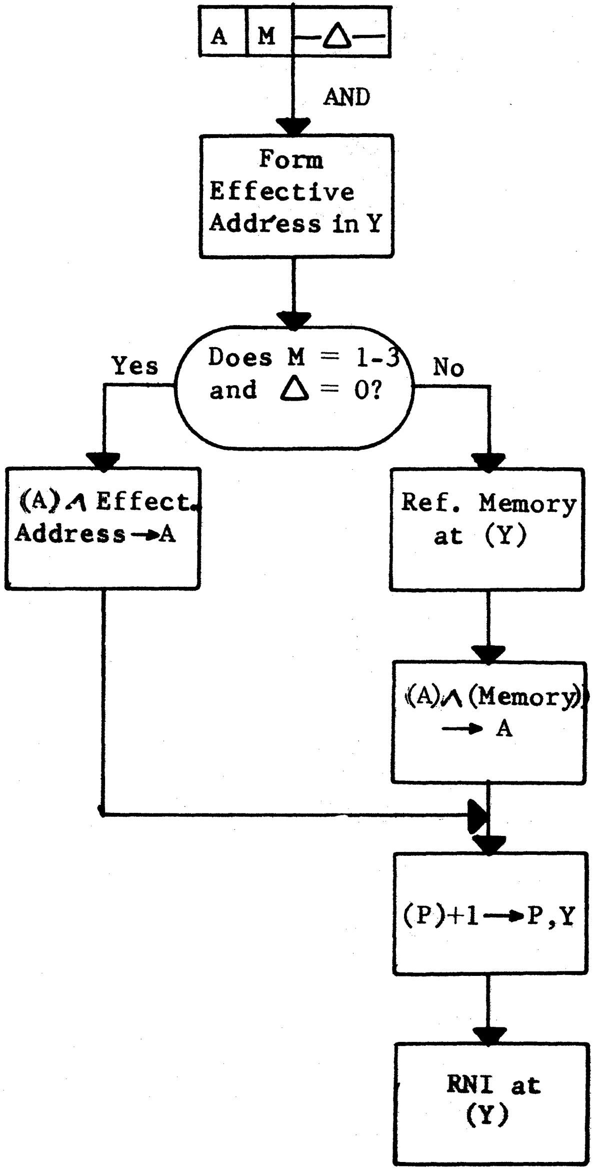

STORE A,
PARITY TO A
7
M
—Δ—
SPA
Form Effective Address in Y
Ref. Memory at (Y)
Does a Protect Fault Exist?
Yes
No
Does a Parity Error Exist?
Yes
No
Is the instruction protected?
No
Yes
Leave (Memory) Unchanged
(A)→ Memory
Clear A
Prot. Bit∨ Parity Bit →A
(P)+1→P,Y& RNI at (Y)


ADD
8
M
—Δ—
ADD
Form Effective Address in Y
Does M = 1-3 and Δ = 0?
Yes
No
A + Effect. Address→A
Ref. Memory at (Y)
(A)+(Memory) →A
Record Overflow Status
(P)+1 →P,Y
RNI at (Y)

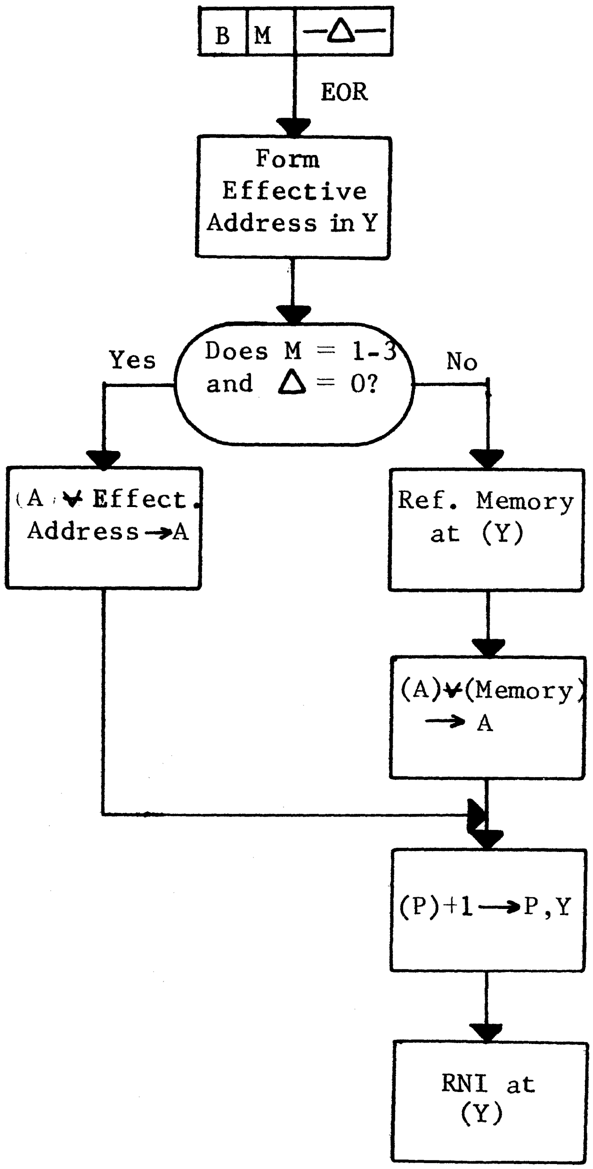
SUBTRACT



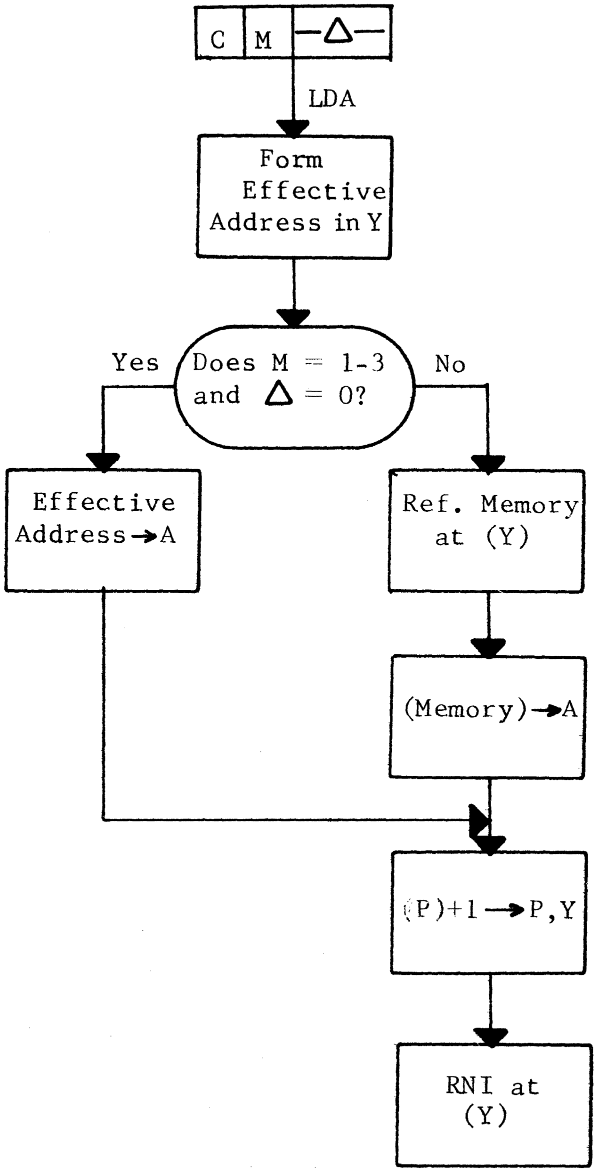
AND WITH A



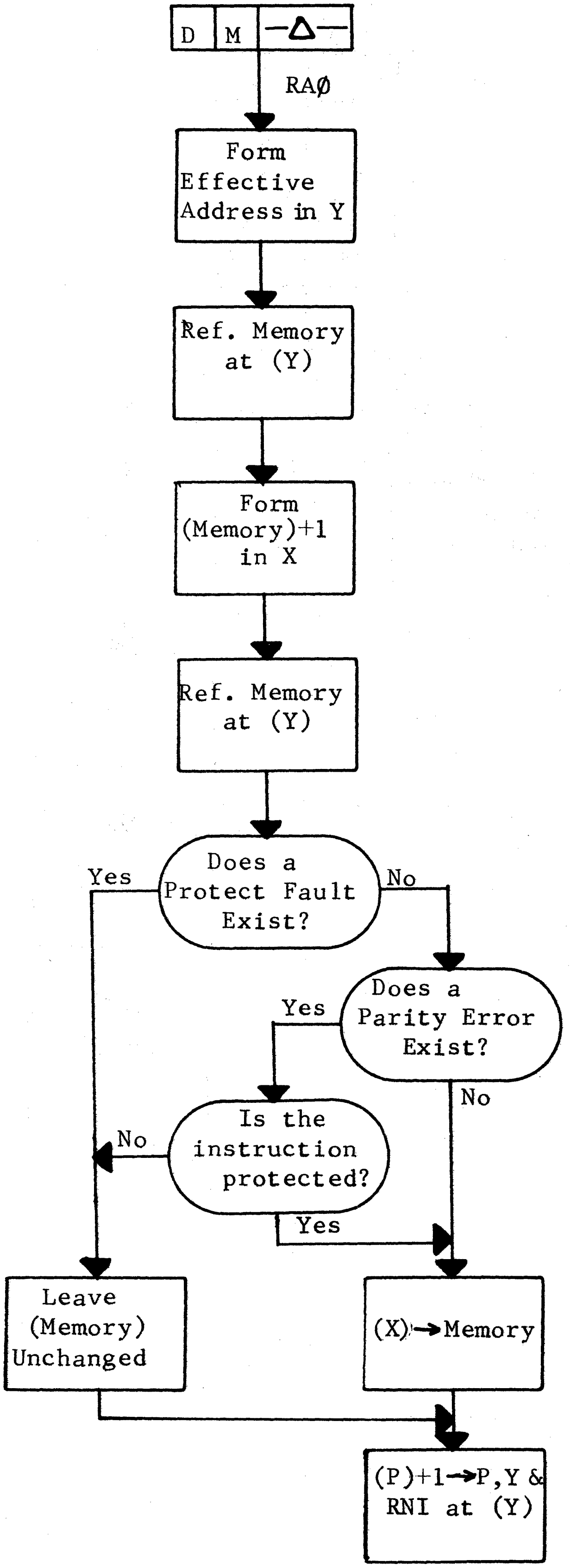
EXCLUSIVE OR
WITH A



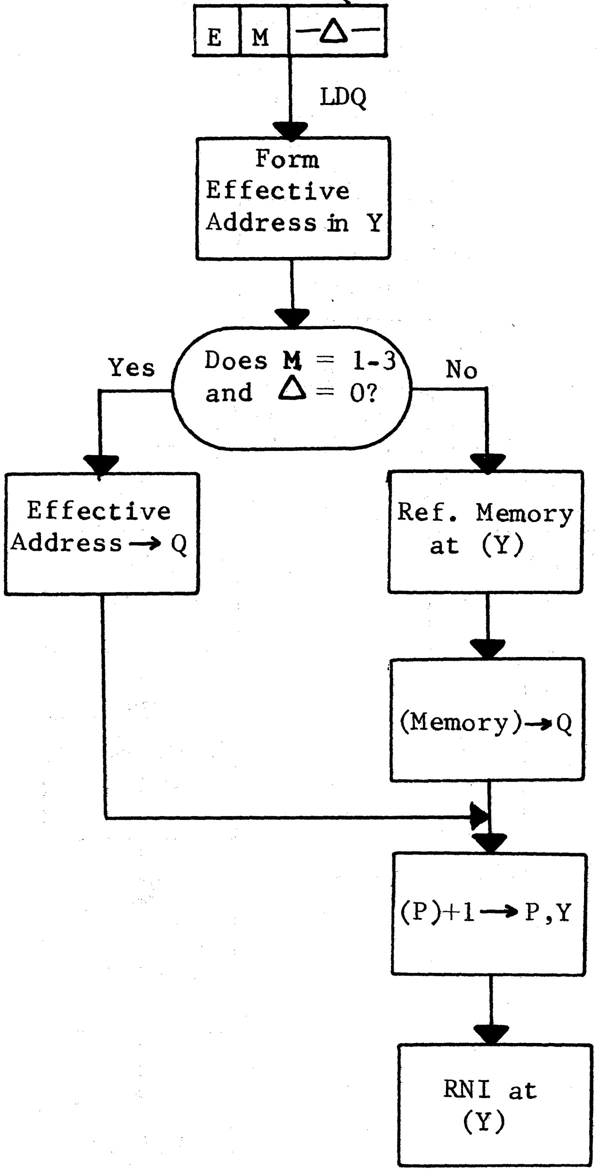
LOAD A

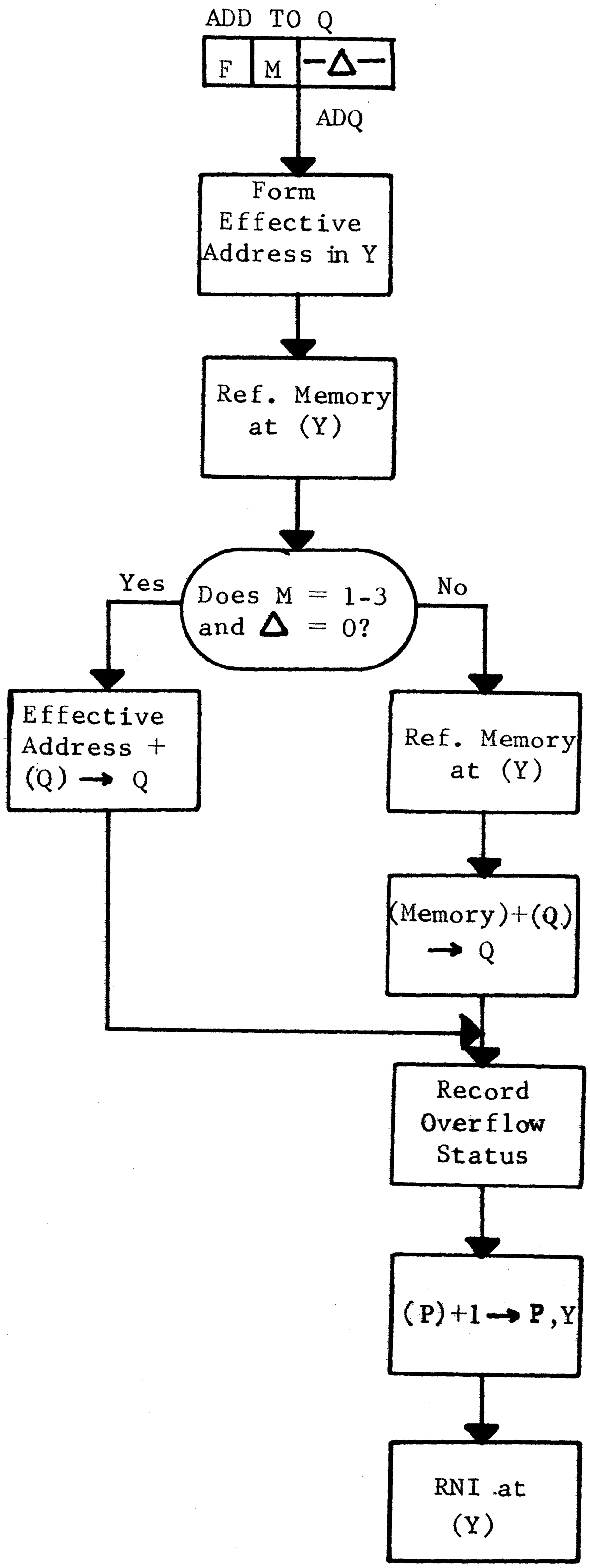


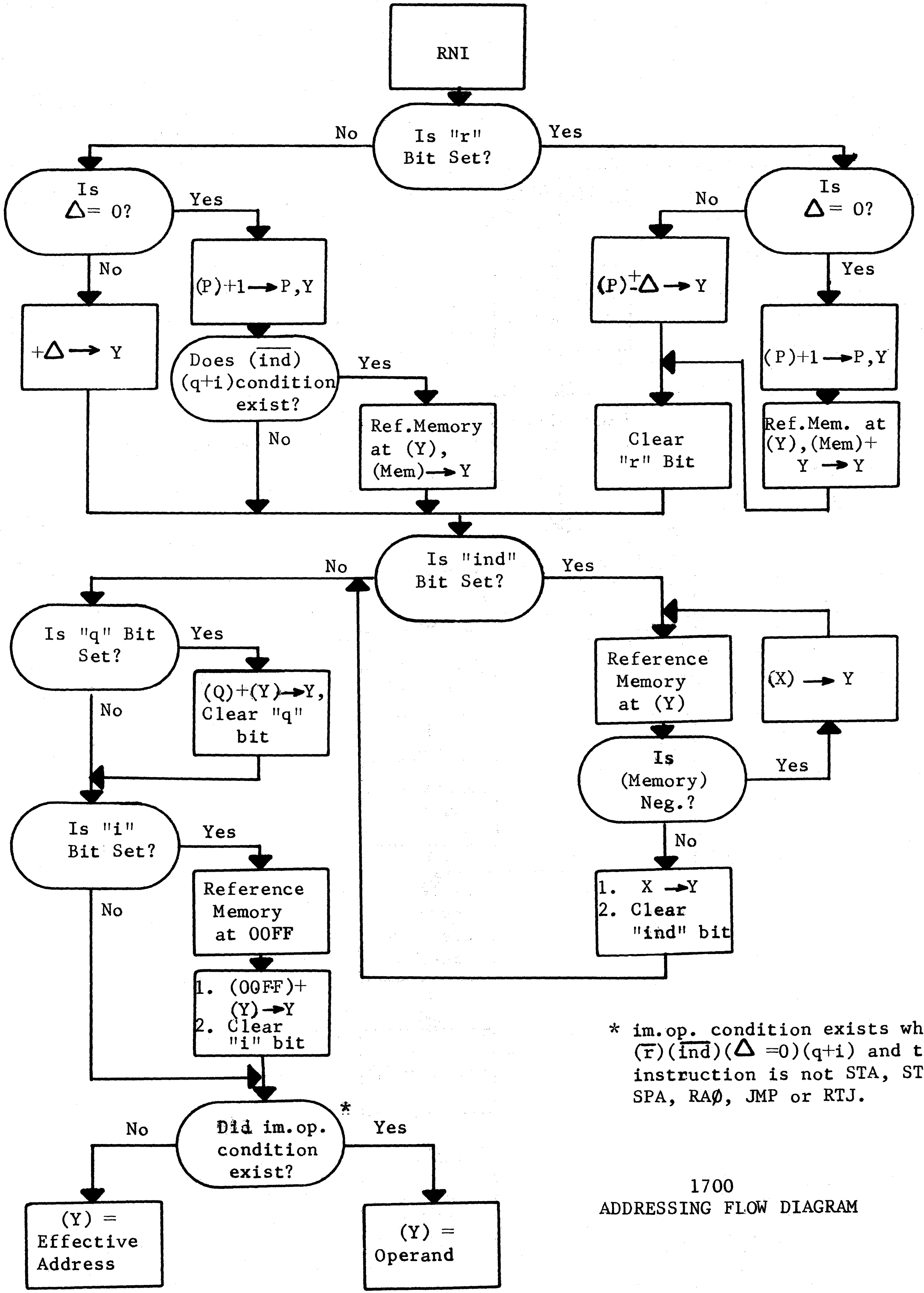
REPLACE ADD ONE



LOAD Q




ADD TO Q
F
M
Δ
ADQ
Form Effective Address in Y
Ref. Memory at (Y)
Does M = 1-3 and Δ = 0?
Yes
No
Effective Address + (Q) → Q
Ref. Memory at (Y)
(Memory)+(Q) → Q
Record Overflow Status
(P)+1 → P,Y
RNI at (Y)



* im.op. condition exists when: $(\overline{r})(\overline{ind})(\Delta=0)(q+i)$ and the instruction is not STA, STQ, SPA, RAØ, JMP or RTJ.

1700
ADDRESSING FLOW DIAGRAM

TABLE I

1700 INSTRUCTIONS (NUMERICAL LISTING)

FORMAT (HEX)				DESCRIPTION	MNE
0	0	▨	▨	SELECTIVE STOP	SLS
0	1	0	S	SKIP IF A IS + ZERO	SAZ
0	1	1	S	SKIP IF A IS NON-ZERO	SAN
0	1	2	S	SKIP IF A IS PLUS	SAP
0	1	3	S	SKIP IF A IS MINUS	SAM
0	1	4	S	SKIP IF Q IS + ZERO	SQZ
0	1	5	S	SKIP IF Q IS NON-ZERO	SQN
0	1	6	S	SKIP IF Q IS PLUS	SQP
0	1	7	S	SKIP IF Q IS MINUS	SQM
0	1	8	S	SKIP IF SKIP SWITCH SET	SWS
0	1	9	S	SKIP IF SKIP SWITCH NOT SET	SWN
0	1	A	S	SKIP IF OVERFLOW FAULT	SØV
0	1	B	S	SKIP IF NO OVERFLOW FAULT	SNØ
0	1	C	S	SKIP IF STORAGE PARITY ERROR	SPE
0	1	D	S	SKIP IF NO STORAGE PARITY ERROR	SNP
0	1	E	S	SKIP IF PROGRAM PROTECT FAULT	SPF
0	1	F	S	SKIP IF NO PROGRAM PROTECT FAULT	SNF
0	2	△		INPUT TO A	INP
0	3	△		OUTPUT FROM A	ØUT
0	4	▨	▨	ENABLE INTERRUPTS (REQUIRES PROTECTION)	EIN
0	5	▨	▨	INHIBIT INTERRUPTS (REQUIRES PROTECTION)	IIN
0	6	▨	▨	SET THE PROGRAM PROTECT BIT (REQUIRES PROTECTION)	SPB
0	7	▨	▨	CLEAR THE PROGRAM PROTECT BIT (REQUIRES PROTECTION)	CPB
0	8	0	0	SUM OF FFFF AND FFFF TO NO PLACE	
0	8	0	1	SUM OF FFFF AND FFFF TO MASK	
0	8	0	2	SUM OF FFFF AND FFFF TO Q	
0	8	0	3	SUM OF FFFF AND FFFF TO Q AND MASK	
0	8	0	4	SUM OF FFFF AND FFFF TO A	
0	8	0	5	SUM OF FFFF AND FFFF TO A AND MASK	
0	8	0	6	SUM OF FFFF AND FFFF TO A AND Q	
0	8	0	7	SUM OF FFFF AND FFFF TO A, Q AND MASK	

▨ MEANS NOT USED

NOTE: Any 08 instruction having bit 00 = "1" requires protection

FORMAT(HEX)				DESCRIPTION	MNE
0	8	0	8	SUM OF FFFF AND (MASK) TO NO PLACE	
0	8	0	9	SUM OF FFFF AND (MASK) TO MASK	
0	8	0	A	SUM OF FFFF AND (MASK) TO Q	
0	8	0	B	SUM OF FFFF AND (MASK) TO Q AND MASK	
0	8	0	C	SUM OF FFFF AND (MASK) TO A	
0	8	0	D	SUM OF FFFF AND (MASK) TO A AND MASK	
0	8	0	E	SUM OF FFFF AND (MASK) TO A AND Q	
0	8	0	F	SUM OF FFFF AND (MASK) TO A, Q AND MASK	
0	8	1	0	SUM OF FFFF AND (Q) TO NO PLACE	
0	8	1	1-7	SUM OF FFFF AND (Q) TO DESTINATION REGISTER (S)	
0	8	1	8	SUM OF FFFF AND (Q)∨(MASK) TO NO PLACE *	
0	8	1	9-F	SUM OF FFFF AND (Q)∨(MASK) TO DESTINATION REGISTER (S) *	
0	8	2	0	SUM OF (A) AND FFFF TO NO PLACE	
0	8	2	1-7	SUM OF (A) AND FFFF TO DESTINATION REGISTER (S)	
0	8	2	8	SUM OF (A) AND (MASK) TO NO PLACE	AAM
0	8	2	9-F	SUM OF (A) AND (MASK) TO DESTINATION REGISTER (S)	AAM
0	8	3	0	SUM OF (A) AND (Q) TO NO PLACE	AAQ
0	8	3	1-7	SUM OF (A) AND (Q) TO DESTINATION REGISTER (S)	AAQ
0	8	3	8	SUM OF (A) AND (Q)∨(MASK) TO NO PLACE *	AAB
0	8	3	9-F	SUM OF (A) AND (Q)∨(MASK) TO DESTINATION REGISTER (S) *	AAB
0	8	4	0	EXCLUSIVE OR OF FFFF AND FFFF TO NO PLACE	CLR
0	8	4	1-7	EXCLUSIVE OR OF FFFF AND FFFF TO DESTINATION REGISTER (S)	CLR
0	8	4	8	EXCLUSIVE OR OF FFFF AND (MASK) TO NO PLACE	TCM
0	8	4	9-F	EXCLUSIVE OR OF FFFF AND (MASK) TO DESTINATION REGISTER (S)	TCM
0	8	5	0	EXCLUSIVE OR OF FFFF AND (Q) TO NO PLACE	TCQ
0	8	5	1-7	EXCLUSIVE OR OF FFFF AND (Q) TO DESTINATION REGISTER (S)	TCQ
0	8	5	8	EXCLUSIVE OR OF FFFF AND (Q)∨(MASK) TO NO PLACE *	TCB
0	8	5	9-F	EXCLUSIVE OR OF FFFF AND (Q)∨(MASK) TO DESTINATION REGISTER(S)*	TCB

* ∨ MEANS INCLUSIVE OR

NOTE: Any 08 instruction having bit 00 = "1" requires protection

FORMAT(HEX)				DESCRIPTION	MNE
0	8	6	0	EXCLUSIVE OR OF (A) AND FFFF TO NO PLACE	TCA
0	8	6	1-7	EXCLUSIVE OR OF (A) AND FFFF TO DESTINATION REGISTEP (S)	TCA
0	8	6	8	EXCLUSIVE OR OF (A) AND (M) TO NO PLACE	EAM
0	8	6	9-F	EXCLUSIVE OR OF (A) AND (M) TO DESTINATION REGISTER (S)	EAM
0	8	7	0	EXCLUSIVE OR OF (A) AND (Q) TO NO PLACE	EAQ
0	8	7	1-7	EXCLUSIVE OR OF (A) AND (Q) TO DESTINATION REGISTER (S)	EAQ
0	8	7	8	EXCLUSIVE OR OF (A) AND (Q)∨(M) TO NO PLACE *	EAB
0	8	7	9-F	EXCLUSIVE OR OF (A) AND (Q)∨(M) TO DESTINATION REGISTER (S)*	EAB
0	8	8	0	LOGICAL PRODUCT OF FFFF AND FFFF TO NO PLACE	SET
0	8	8	1-7	LOGICAL PRODUCT OF FFFF AND FFFF TO DESTINATION REGISTER (S)	SET
0	8	8	8	LOGICAL PRODUCT OF FFFF AND (MASK) TO NO PLACE	TRM
0	8	8	9-F	LOGICAL PRODUCT OF FFFF AND (MASK) TO DESTINATION REGISTER(S)	TRM
0	8	9	0	LOGICAL PRODUCT OF FFFF AND (Q) TO NO PLACE	TRQ
0	8	9	1-7	LOGICAL PRODUCT OF FFFF AND (Q) TO DESTINATION REGISTER (S)	TRQ
0	8	9	8	LOGICAL PRODUCT OF FFFF AND (Q)∨(M) TO NO PLACE *	TRB
0	8	9	9-F	LOGICAL PRODUCT OF FFFF AND (Q)∨(M) TO DESTINATION REG. (S)*	TRB
0	8	A	0	LOGICAL PRODUCT OF (A) AND FFFF TO NO PLACE	TRA
0	8	A	1-7	LOGICAL PRODUCT OF (A) AND FFFF TO DESTINATION REGISTER (S)	TRA
0	8	A	8	LOGICAL PRODUCT OF (A) AND (MASK) TO NO PLACE	LAM
0	8	A	9-F	LOGICAL PRODUCT OF (A) AND (MASK) TO DESTINATION REGISTER (S)	LAM
0	8	B	0	LOGICAL PRODUCT OF (A) AND (Q) TO NO PLACE	LAQ
0	8	B	1-7	LOGICAL PRODUCT OF (A) AND (Q) TO DESTINATION REGISTER (S)	LAQ
0	8	B	8	LOGICAL PRODUCT OF (A) AND (Q)∨(MASK) TO NO PLACE *	LAB
0	8	B	9-F	LOGICAL PRODUCT OF (A) AND (Q)∨(MASK) TO DESTINATION R. (S)*	LAB
0	8	C	0	COMPLEMENTED LOG. PROD. OF FFFF AND FFFF TO NO PLACE	
0	8	C	1-7	COMP. LOG. PROD. OF FFFF AND FFFF TO DESTINATION REG. (S)	
0	8	C	8	COMPLEMENTED LOG. PROD. OF FFFF AND (MASK) TO NO PLACE	
0	8	C	9-F	COMP. LOG. PROD. OF FFFF AND (MASK) TO DESTINATION REG. (S)	

* ∨ MEANS INCLUSIVE OR

NOTE: Any 08 instruction having bit 00 = "1" requires protection

FORMAT (HEX)				DESCRIPTION	MNE
0	8	D	0	COMPLEMENTED LOG. PROD. OF FFFF and (Q)→ NO PLACE	
0	8	D	1-7	COMPLEMENTED LOG. PROD. OF FFFF and (Q)→ DESTINATION REGISTER(s)	
0	8	D	8	COMPLEMENTED LOG. PROD. OF FFFF and (Q)∨(Mask)→No Place *	
0	8	D	9-F	COMPLEMENTED LOG. PROD. OF FFFF and (Q)∨(Mask)→DESTINATION REGISTER(s)*	
0	8	E	0	COMPLEMENTED LOG. PROD. OF (A) and FFFF→NO PLACE	
0	8	E	1-7	COMPLEMENTED LOG. PROD. OF (A) and FFFF→DESTINATION REGISTER(s)	
0	8	E	8	COMPLEMENTED LOG. PROD. OF (A) and (Mask)→NO PLACE	CAM
0	8	E	9-F	COMPLEMENTED LOG. PROD. OF (A) and (Mask)→DESTINATION REGISTER(s)	CAM
0	8	F	0	COMPLEMENTED LOG. PROD. OF (A) and (Q)→ NO PLACE	CAQ
0	8	F	1-7	COMPLEMENTED LOG. PROD. OF (A) and (Q)→ DESTINATION REGISTER(s)	CAQ
0	8	F	8	COMPLEMENTED LOG. PROD. OF (A) and (Q)∨(Mask)→ NO PLACE*	CAB
0	8	F	9-F	COMPLEMENTED LOG. PROD. OF (A) and (Q)∨(Mask)→DESTINATION REGISTER(s)*	CAB
0	9	-Δ-		INCREASE A	INA
0	A	-Δ-		ENTER A	ENA
0	B	▨		NO OPERATION	
0	C	-Δ-		ENTER Q	ENQ
0	D	-Δ-		INCREASE Q	INQ
0	E	-Δ-		EXIT THE INTERRUPT STATE (REQUIRES PROTECTION)	EXI
0	F	0/1	0-F	NO OPERATION	NOP
0	F	2/3	0-F	Q RIGHT SHIFT	QRS
0	F	4/5	0-F	A RIGHT SHIFT	ARS
0	F	6/7	0-F	LONG RIGHT SHIFT	LRS
0	F	8/9	0-F	NO OPERATION	
0	F	A/B	0-F	Q LEFT SHIFT	QLS
0	F	C/D	0-F	A LEFT SHIFT	ALS
0	F	E/F	0-F	LONG LEFT SHIFT	LLS

* MEANS INCLUSIVE OR

▨ MEANS NOT USED

NOTE: Any 08 instruction having bit 00 = "1" requires protection

FORMAT (HEX)			DESCRIPTION	MNE
1	*	-Δ-	JUMP	JMP
2	*	-Δ-	MULTIPLY INTEGER	MUI
3	*	-Δ-	DIVIDE INTEGER	DVI
4	*	-Δ-	STORE Q	STQ
5	*	-Δ-	RETURN JUMP	RTJ
6	*	-Δ-	STORE A	STA
7	*	-Δ-	STORE A, SEND PARITY TO A_{00}	SPA
8	*	-Δ-	ADD TO A	ADD
9	*	-Δ-	SUBTRACT FROM A	SUB
A	*	-Δ-	AND WITH A	AND
B	*	-Δ-	EXCLUSIVE OR WITH A	EØR
C	*	-Δ-	LOAD A	LDA
D	*	-Δ-	REPLACE ADD ONE IN STORAGE	RAØ
E	*	-Δ-	LOAD Q	LDQ
F	*	-Δ-	ADD TO Q	ADQ

* Addressing Code =

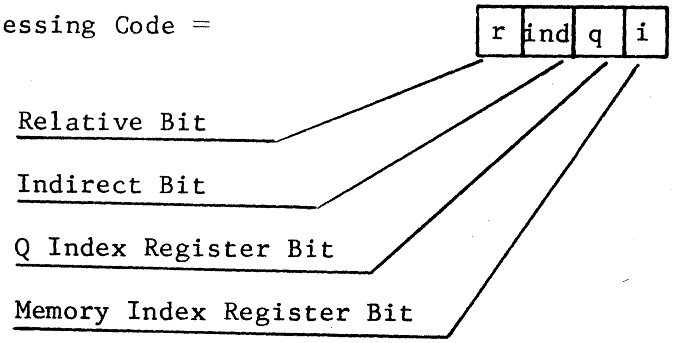


TABLE II

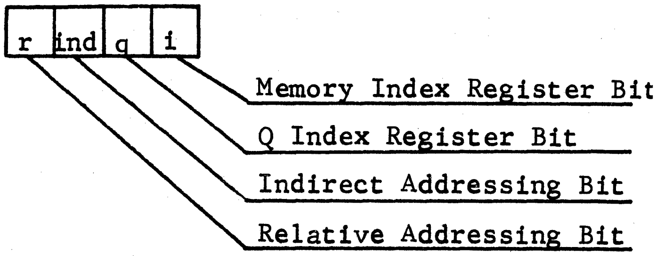
1700 INSTRUCTIONS (ALPHABETICAL LISTING)

MNE	DESCRIPTION	FORMAT (HEX)			
AAB	TRANSFER ARITHMETIC SUM OF A AND Q$\vee$M**	0	8	3	8-F
AAM	TRANSFER ARITHMETIC SUM OF A AND M	0	8	2	8-F
AAQ	TRANSFER ARITHMETIC SUM OF A AND Q	0	8	3	0-7
ADD	ADD TO A	8	*	-Δ-	
ADQ	ADD TO Q	F	*	-Δ-	
ALS	A LEFT SHIFT	0	F	C/D	0-F
AND	AND WITH A	A	*	-Δ-	
ARS	A RIGHT SHIFT	0	F	4/5	0-F
CAB	TRANSFER THE COMPLEMENTED LOGICAL PRODUCT OF A AND Q$\vee$M **	0	8	F	8-F
CAM	TRANSFER THE COMPLEMENTED LOGICAL PRODUCT OF A AND M	0	8	E	8-F
CAQ	TRANSFER THE COMPLEMENTED LOGICAL PRODUCT OF A AND Q	0	8	F	0-7
CLR	CLEAR THE CONTENTS OF THE DESTINATION REGISTER(S) TO +ZERO	0	8	4	0-7
CPB	CLEAR THE PROGRAM PROTECT BIT	0	7	▨	
DVI	DIVIDE INTEGER	3	*	-Δ-	
EAB	TRANSFER THE EXCLUSIVE OR OF A AND Q$\vee$M **	0	8	7	8-F
EAM	TRANSFER THE EXCLUSIVE OR OF A AND M	0	8	6	8-F
EAQ	TRANSFER THE EXCLUSIVE OR OF A AND Q	0	8	7	0-7
EIN	ENABLE INTERRUPTS	0	4	▨	
ENA	ENTER A	0	A	-Δ-	
ENQ	ENTER Q	0	C	-Δ-	
EØR	EXCLUSIVE OR WITH A	B	*	-Δ-	
EXI	EXIT THE INTERRUPT STATE	0	E	-Δ-	
IIN	INHIBIT INTERRUPTS	0	5	▨	
INA	INCREASE A	0	9	-Δ-	
INP	INPUT TO A	0	2	-Δ-	
INQ	INCREASE Q	0	D	-Δ-	
JMP	JUMP	1	*	-Δ-	
LAB	TRANSFER THE LOGICAL PRODUCT OF A AND Q$\vee$M **	0	8	B	8-F
LAM	TRANSFER THE LOGICAL PRODUCT OF A AND M	0	8	A	8-F
LAQ	TRANSFER THE LOGICAL PRODUCT OF A AND Q	0	8	B	0-7
LDA	LOAD A	C	*	-Δ-	
LDQ	LOAD Q	E	*	-Δ-	
LLS	LONG LEFT SHIFT	0	F	E/F	0-F
LRS	LONG RIGHT SHIFT	0	F	6/7	0-F

M N E	DESCRIPTION	FORMAT (HEX.)			
MUI	MULTIPLY INTEGER	2	*	-Δ-	
NØP	NO OPERATION	0	F	0/1	0-F
ØUT	OUTPUT FROM A	0	3	-Δ-	
QLS	Q LEFT SHIFT	0	F	A/B	0-F
QRS	Q RIGHT SHIFT	0	F	2/3	0-F
RAØ	REPLACE ADD ONE IN STORAGE	D	*	-Δ-	
RTJ	RETURN JUMP	5	*	-Δ-	
SAM	SKIP IF A IS MINUS	0	1	3	S
SAN	SKIP IF A IS NON-ZERO	0	1	1	S
SAP	SKIP IF A IS PLUS	0	1	2	S
SAZ	SKIP IF A IS ZERO	0	1	0	S
SET	SET THE CONTENTS OF THE DESTINATION REG.(S) TO ALL ONES	0	8	8	0-7
SLS	SELECTIVE STOP	0	0	▨	
SNØ	SKIP IF NO OVERFLOW	0	1	B	S
SNF	SKIP IF NO PROGRAM PROTECT FAULT	0	1	F	S
SNP	SKIP IF NO STORAGE PARITY ERROR	0	1	D	S
SØV	SKIP IF OVERFLOW	0	1	A	S
SPA	STORE A, SEND PARITY BIT TO A_{00}	7	*	-Δ-	
SPB	SET THE PROGRAM PROTECT BIT	0	6	▨	
SPE	SKIP IF STORAGE PARITY ERROR	0	1	C	S
SPF	SKIP IF PROGRAM PROTECT FAULT	0	1	E	S
SQM	SKIP IF Q IS MINUS	0	1	7	S
SQN	SKIP IF Q IS NON-ZERO	0	1	5	S
SQP	SKIP IF Q IS PLUS	0	1	6	S
SQZ	SKIP IF Q IS ZERO	0	1	4	S
STA	STORE A	6	*	-Δ-	
STQ	STORE Q	4	*	-Δ-	
SUB	SUBTRACT FROM A	9	*	-Δ-	
SWN	SKIP IF SKIP SWITCH NOT SET	0	1	9	S
SWS	SKIP IF SKIP SWITCH SET	0	1	8	S

MNE	DESCRIPTION	FORMAT (HEX.)			
TCA	TRANSFER THE COMPLEMENT OF A	0	8	6	0-7
TCB	TRANSFER THE COMPLEMENT OF Q∨M**	0	8	5	8-F
TCM	TRANSFER THE COMPLEMENT OF M	0	8	4	8-F
TCQ	TRANSFER THE COMPLEMENT OF Q	0	8	5	0-7
TRA	TRANSFER A	0	8	A	0-7
TRB	TRANSFER Q∨M**	0	8	9	8-F
TRM	TRANSFER M	0	8	8	8-F
TRQ	TRANSFER Q	0	8	9	0-7

* Addressing Code =



** ∨ means "inclusive or"

▨ means "not used"

TABLE III

1700 ADDRESSING MODES

ADDRESSING CODE	Δ	EFFECTIVE ADDRESS	NEXT INSTRUCTION
0	+0	P + 1	P + 2
1		(P + 1) + (00FF) *	P + 2
2		(P + 1) + (Q) *	P + 2
3		(P + 1) + (Q) + (00FF) *	P + 2
4		(P + 1)	P + 2
5		(P + 1) + (00FF)	P + 2
6		(P + 1) + (Q)	P + 2
7		(P + 1) + (Q) + (00FF)	P + 2
8		P + 1 + (P + 1)	P + 2
9		P + 1 + (P + 1) + (00FF)	P + 2
A		P + 1 + (P + 1) + (Q)	P + 2
B		P + 1 + (P + 1) + (Q) + (00FF)	P + 2
C		(P + 1 + (P + 1))	P + 2
D		(P + 1 + (P + 1)) + (00FF)	P + 2
E		(P + 1 + (P + 1)) + (Q)	P + 2
F	+0	(P + 1 + (P + 1)) + (Q) + (00FF)	P + 2
0	≠0	+Δ	P + 1
1		+Δ + (00FF)	P + 1
2		+Δ + (Q)	P + 1
3		+Δ + (Q) + (00FF)	P + 1
4		(+Δ)	P + 1
5		(+Δ) + (00FF)	P + 1
6		(+ Δ) + (Q)	P + 1
7		(+ Δ) + (Q) + (00FF)	P + 1
8		(P) ±Δ	P + 1
9		(P) ±Δ + (00FF)	P + 1
A		(P) ±Δ + (Q)	P + 1
B		(P) ±Δ + (Q) + (00FF)	P + 1
C		((P) ±Δ)	P + 1
D		((P) ±Δ) + (00FF)	P + 1
E		((P) ±Δ) + (Q)	P + 1
F	≠0	((P) ±Δ) + (Q) + (00FF)	P + 1

* Effective Address = Operand For Instructions Which Read Operands

TABLE IV

HEXADECIMAL ADDITION

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11
3	3	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12
4	4	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13
5	5	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14
6	6	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15
7	7	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16
8	8	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17
9	9	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18
A	A	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19
B	B	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A
C	C	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B
D	D	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C
E	E	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D
F	F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E

HEX	BINARY
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

TABLE V

HEXADECIMAL MULTIPLICATION

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
2	00	02	04	06	08	0A	0C	0E	10	12	14	16	18	1A	1C	1E
3	00	03	06	09	0C	0F	12	15	18	1B	1E	21	24	27	2A	2D
4	00	04	08	0C	10	14	18	1C	20	24	28	2C	30	34	38	3C
5	00	05	0A	0F	14	19	1E	23	28	2D	32	37	3C	41	46	4B
6	00	06	0C	12	18	1E	24	2A	30	36	3C	42	48	4E	54	5A
7	00	07	0E	15	1C	23	2A	31	38	3F	46	4D	54	5B	62	69
8	00	08	10	18	20	28	30	38	40	48	50	58	60	68	70	78
9	00	09	12	1B	24	2D	36	3F	48	51	5A	63	6C	75	7E	87
A	00	0A	14	1E	28	32	3C	46	50	5A	64	6E	78	82	8C	96
B	00	0B	16	21	2C	37	42	4D	58	63	6E	79	84	8F	9A	A5
C	00	0C	18	24	30	3C	48	54	60	6C	78	84	90	9C	A8	B4
D	00	0D	1A	27	34	41	4E	5B	68	75	82	8F	9C	A9	B6	C3
E	00	0E	1C	2A	38	46	54	62	70	7E	8C	9A	A8	B6	C4	D2
F	00	0F	1E	2D	3C	4B	5A	69	78	87	96	A5	B4	C3	D2	E1

HEX	BINARY
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

TABLE VI

CONVERSION

HEX.	DEC.	HEX.	DEC.	HEX.	DEC.	HEX.	DEC.	HEX.	DEC.
1	1	10	16	100	256	1000	4096	10000	65536
2	2	20	32	200	512	2000	8192	20000	131072
3	3	30	48	300	768	3000	12288	30000	196608
4	4	40	64	400	1024	4000	16384	40000	262144
5	5	50	80	500	1280	5000	20480	50000	327680
6	6	60	96	600	1536	6000	24576	60000	393216
7	7	70	112	700	1792	7000	28672	70000	458752
8	8	80	128	800	2048	8000	32768	80000	524288
9	9	90	144	900	2304	9000	36864	90000	589824
A	10	A0	160	A00	2560	A000	40960	A0000	655360
B	11	B0	176	B00	2816	B000	45056	B0000	720896
C	12	C0	192	C00	3072	C000	49152	C0000	786432
D	13	D0	208	D00	3328	D000	53248	D0000	851968
E	14	E0	224	E00	3584	E000	57344	E0000	917504
F	15	F0	240	F00	3840	F000	61440	F0000	983040

HEX.	DEC.	HEX.	DEC.	HEX.	DEC.
100000	1048576	1000000	16777216	10000000	268435456
200000	2097152	2000000	33554432	20000000	536870912
300000	3145728	3000000	50331648	30000000	805306368
400000	4194304	4000000	67108864	40000000	1073741824
500000	5242880	5000000	83886080	50000000	1342177280
600000	6291456	6000000	100663296	60000000	1610612736
700000	7340032	7000000	117440512	70000000	1879048192
800000	8388608	8000000	134217728	80000000	2147483648
900000	9437184	9000000	150994944	90000000	2415919104
A00000	10485760	A000000	167772160	A0000000	2684354560
B00000	11534336	B000000	184549376	B0000000	2952790016
C00000	12582912	C000000	201326592	C0000000	3221225472
D00000	13631488	D000000	218103808	D0000000	3489660928
E00000	14680064	E000000	234881024	E0000000	3758096384
F00000	15728640	F000000	251658240	F0000000	4026531840

POWERS OF SIXTEEN	
16^0	1
16^1	16
16^2	256
16^3	4096
16^4	65536
16^5	1048576
16^6	16777216
16^7	268435456

TABLE VII

LOGICAL PRODUCT
($\wedge$)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
2	0	0	2	2	0	0	2	2	0	0	2	2	0	0	2	2
3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	0	0	0	0	4	4	4	4	0	0	0	0	4	4	4	4
5	0	1	0	1	4	5	4	5	0	1	0	1	4	5	4	5
6	0	0	2	2	4	4	6	6	0	0	2	2	4	4	6	6
7	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
8	0	0	0	0	0	0	0	0	8	8	8	8	8	8	8	8
9	0	1	0	1	0	1	0	1	8	9	8	9	8	9	8	9
A	0	0	2	2	0	0	2	2	8	8	A	A	8	8	A	A
B	0	1	2	3	0	1	2	3	8	9	A	B	8	9	A	B
C	0	0	0	0	4	4	4	4	8	8	8	8	C	C	C	C
D	0	1	0	1	4	5	4	5	8	9	8	9	C	D	C	D
E	0	0	2	2	4	4	6	6	8	8	A	A	C	C	E	E
F	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

EXCLUSIVE OR
($\veebar$)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	0	3	2	5	4	7	6	9	8	B	A	D	C	F	E
2	2	3	0	1	6	7	4	5	A	B	8	9	E	F	C	D
3	3	2	1	0	7	6	5	4	B	A	9	8	F	E	D	C
4	4	5	6	7	0	1	2	3	C	D	E	F	8	9	A	B
5	5	4	7	6	1	0	3	2	D	C	F	E	9	8	B	A
6	6	7	4	5	2	3	0	1	E	F	C	D	A	B	8	9
7	7	6	5	4	3	2	1	0	F	E	D	C	B	A	9	8
8	8	9	A	B	C	D	E	F	0	1	2	3	4	5	6	7
9	9	8	B	A	D	C	F	E	1	0	3	2	5	4	7	6
A	A	B	8	9	E	F	C	D	2	3	0	1	6	7	4	5
B	B	A	9	8	F	E	D	C	3	2	1	0	7	6	5	4
C	C	D	E	F	8	9	A	B	4	5	6	7	0	1	2	3
D	D	C	F	E	9	8	B	A	5	4	7	6	1	0	3	2
E	E	F	C	D	A	B	8	9	6	7	4	5	2	3	0	1
F	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

INCLUSIVE OR
($\vee$)

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	1	3	3	5	5	7	7	9	9	B	B	D	D	F	F
2	2	3	2	3	6	7	6	7	A	B	A	B	E	F	E	F
3	3	3	3	3	7	7	7	7	B	B	B	B	F	F	F	F
4	4	5	6	7	4	5	6	7	C	D	E	F	C	D	E	F
5	5	5	7	7	5	5	7	7	D	D	F	F	D	D	F	F
6	6	7	6	7	6	7	6	7	E	F	E	F	E	F	E	F
7	7	7	7	7	7	7	7	7	F	F	F	F	F	F	F	F
8	8	9	A	B	C	D	E	F	8	9	A	B	C	D	E	F
9	9	9	B	B	D	D	F	F	9	9	B	B	D	D	F	F
A	A	B	A	B	E	F	E	F	A	B	A	B	E	F	E	F
B	B	B	B	B	F	F	F	F	B	B	B	B	F	F	F	F
C	C	D	E	F	C	D	E	F	C	D	E	F	C	D	E	F
D	D	D	F	F	D	D	F	F	D	D	F	F	D	D	F	F
E	E	F	E	F	E	F	E	F	E	F	E	F	E	F	E	F
F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F

COMMENT SHEET

1700 COMPUTER
MAINTENANCE TRAINING MANUAL, VOLUME I

Publication Number 60169500A

FROM: Name: ______________________________

Address: ______________________________

COMMENTS: (Describe errors, suggested additions or deletions, and include page numbers, etc.)

CONTROL DATA
CORPORATION