

# IBM

## IBM 7040 and 7044 Data Processing Systems

Student Text



International Business Machines Corporation  
Data Processing Division  
112 East Post Road, White Plains, New York



**Student Text**

**IBM 7040 and 7044 Data Processing Systems**

## Preface

The primary aim of this publication is to instruct novice 7040/7044 programmers. The material may be used with more experienced programmers by skipping sections familiar to the student and stressing the new and unfamiliar. Material should be presented serially; each section requires understanding of the previous one.

The program examples and techniques use symbolic language to emphasize programming concepts rather than machine details. Problems for the student are included in most sections; answers are in the Appendix.

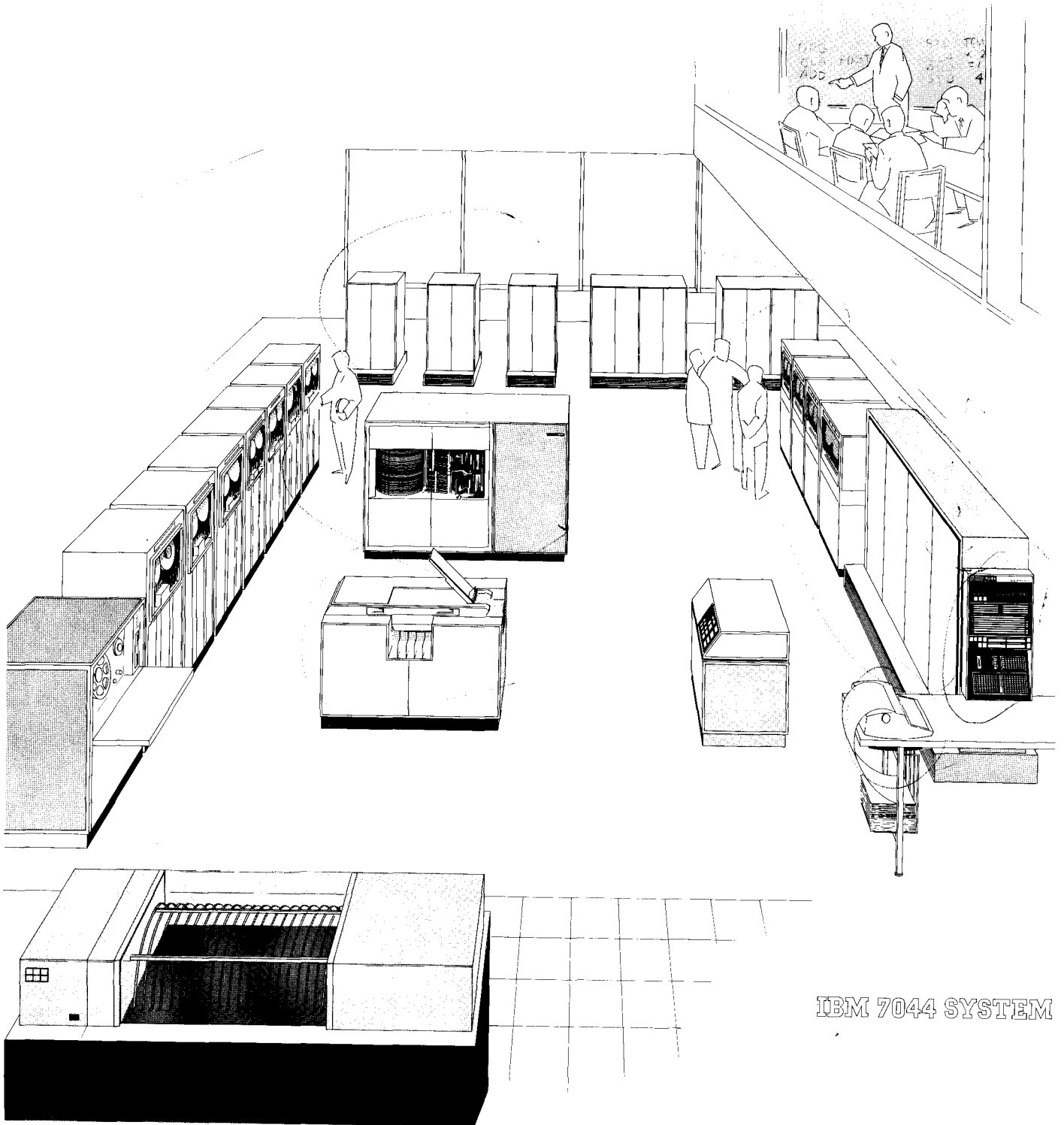
### MINOR REVISION (May 1963)

This edition, Form C22-6732-1, is a minor revision of, and obsoletes, the preceding edition, Form C22-6732.

Copies of this and other IBM publications can be obtained through IBM Branch Offices. Address comments concerning the content of this publication to:  
IBM Corporation, Customer Manuals, Dept. B98, PO Box 390, Poughkeepsie, N. Y.

## Contents

<b>Introduction</b> .....	5	Devices and Control Units used on 7040/7044 Systems .....	81
<b>Computer Data and Instructions</b> .....	14	Data Channels .....	83
Numbers Concept .....	14	<b>IBM 1414 Input/Output Synchronizers</b> .....	87
Number Conversions .....	18	IBM 1414 Models 1, 2, and 7 .....	87
Computer Codes .....	20	IBM 1414 Models 3, 4, and 5 .....	97
Code Definitions .....	21	<b>Punched Cards, Readers, Punches, and Printers</b> ...	99
Processing Unit Operations .....	23	IBM 1403 Printer .....	105
<b>Introduction to Programming Systems</b> .....	28	Console Typewriter .....	106
COBOL System .....	30	<b>Disk Storage and Other Optional Features</b> .....	108
FORTRAN System .....	32	IBM 1301 and IBM 7631 .....	108
Program Checkout .....	32	Direct Data Connection .....	112
Input/Output Control Systems .....	33	Storage Protection Instructions .....	114
<b>IBM 7040/7044 Programming Systems Programs</b> ..	36	IBM 1401 Data Processing System .....	114
Over-all Operation .....	37	<b>Trapping</b> .....	116
Macro Assembly Program .....	38	Processing Unit Traps .....	116
Macro Assembly Program Language .....	39	Data Channel Traps .....	119
<b>Instruction Descriptions and Use</b> .....	43	Trap Flow Chart .....	120
Instruction Specifications .....	43	Instructions used with Trapping .....	122
Fixed-Point Arithmetic Instructions .....	44	<b>Systems Compatibility</b> .....	124
Shifting Operations .....	49	Compatible Features .....	124
Control Instructions .....	52	Incompatible Features .....	124
Indexing Operations .....	55	Detailed Compatibility Information .....	124
Logic Operations .....	62	Programming Compatibility Notes .....	126
Packing and Unpacking .....	64	Trapping Notes .....	127
Character Handling Operations .....	67	<b>Programming Examples</b> .....	128
Data Transmission .....	67	<b>Appendix</b> .....	139
Floating-Point Operations .....	68	A. Instructions .....	139
Single-Precision Floating-Point Instructions ..	71	B. Instruction List—Alphabetic Order with Formats ..	142
Trapping .....	73	C. Powers of Two Table .....	147
Double-Precision Floating-Point Instructions ..	74	D. Octal-Decimal Integer Conversion Table .....	148
<b>IBM 7040/7044 Input/Output Control System</b> ..	76	E. Octal-Decimal Fraction Conversion Table .....	152
Basic Concepts .....	76	F. Scaling for Fixed-Point Calculation .....	155
IOCS Organization .....	77	G. Problem Answers .....	159
<b>Input/Output Devices and Operations</b> .....	79	<b>Index</b> .....	167
Reading and Writing .....	79		
Data Buffering .....	79		



IBM 7044 SYSTEM

## IBM 7040-7044 Data Processing Systems Student Text

Data processing consists of planned actions and operations upon data to produce a desired result. These actions and operations are accomplished with a data processing system — a combination of units that normally includes input, storage, processing, and output devices. The systems are designed to handle business and scientific data at electronic speeds with internal checks for accuracy and have as their key element a high-speed computer — the processing unit.

Data processing systems vary in size, ability, speed, and cost but, regardless of the information to be processed or the equipment used, all systems involve at least three basic considerations:

1. The source data or *input* entering the system.
2. The planned *processing* within the system.
3. The end result or *output* from the system.

*Input Data* may be classified into two basic groups. The first, historical data, is a record of something that has already occurred. The second, real-time data, originates as something happens.

*Processing* is carried out in a pre-established sequence of instructions, which is automatically followed by the computer. The plan of processing is always of human origin. By calculation, sorting, analysis, and other operations, the computer arrives at a result, which may be used for further processing or control or may be recorded as output.

*Output* from the computer may take the form of printed reports, punched cards, reels of magnetic tape or paper tape, messages on communication networks, or any combination of these forms. Output may be used to directly control other devices or processes.

### Stored Program Concepts

After data are received as input, the data processing system can take over the complete processing and preparation of results; however, all procedural steps that are to take place within the computer system must be precisely defined in terms of operations the system can perform. The definitions of these procedural steps are called instructions.

A series of instructions pertaining to an entire procedure is a program. In current data processing systems, the program is stored internally, and the system has electronic-speed access to the instructions in this stored program.

All instructions and data words are assigned a number as they are placed in core storage. This number

is called an address and corresponds to a specific core storage location. Using the address, the program can locate and retrieve the information as needed during processing.

### Instructions

Each computer operation is directed by an instruction — a unit of specific information located in core storage. The processing unit interprets this information as an operation to be performed. If data are involved, the instruction directs the computer to the data. If some device — a magnetic tape unit for example — is to be controlled, the instruction specifies the device and the required operation.

Instructions may shift data from one location in storage to another, they may cause a tape unit to rewind, they may change the condition of an indicator, or they may change the contents of a register or counter. Some instructions arbitrarily, or as a result of some machine or data indication, can specify the storage location of the next instruction or block of instructions to be performed.

Most instructions consist of at least two parts (Figure 1):

*The Operation Part* designates read, write, add, subtract, compare, move data, and so on.

*The Operand* designates the address of the data or device needed by the operation part. Operands are also used to designate the number of places the contents of a register are to be shifted, to set an indicator, to test an indicator, and so on.

During an instruction cycle, an instruction is removed from storage and analyzed by the processing unit. Each computer operation, such as add or divide, is assigned a unique code, which can be recognized

Operation Part	Operand Part
Read Select	Select a tape unit for reading and read one record into storage locations 1000 through 1050
Clear and Add	Quantity in storage location 1004 is placed in the accumulator register. This action clears old data from the accumulator.
Subtract	Quantity in storage location 1005 from the contents of the accumulator register.
Store	Result in storage location 1051
Transfer	To instruction in storage location 5004

Figure 1. Instruction Format

by the computer. The operand further defines the function of the operation – for example: to perform arithmetic, the storage location of one of the factors involved is indicated; for input or output devices, the unit to be used is specified; for reading or writing, the area in storage in which the data will be located is indicated.

Because instructions are stored in the same storage medium as data are, they must be represented in the same form as data. The number of storage positions required by a single instruction is usually constant for a given computer; stated another way, instructions are usually fixed in length.

In general, no particular areas of storage are reserved for instructions only. In most instances, they are grouped and placed in ascending sequential locations in the normal order in which they will be executed by the computer. The order of execution may be varied, however, by special instructions or recognition of certain conditions within the system.

The normal sequence of computer operation in a complete program is:

1. The computer locates and executes the first instruction.
2. The computer locates and executes the next instruction.
3. The process continues automatically, instruction by instruction, until the program is completed or until the computer is instructed to stop.

### Serial and Parallel Operation

Computers are classified as either serial or parallel, depending on the method the computer uses to perform arithmetic.

In a serial computer, numbers to be added are considered one position at a time (the units position, tens position, hundreds, and so on) in the same way that addition is done with paper and pencil. Whenever a carry is developed, it is retained temporarily and, on the next machine cycle, is added to the sum of the next higher-order position.

The time required for serial operation depends on the number of digits in the factors to be added. Figure 2 shows serial addition.

	First Step	Second Step	Third Step	Fourth Step
Addend	1234	1234	1234	1234
Augend	2459	2459	2459	2459
Carry	1	1		
Sum	3	93	693	3693

Figure 2. Serial Addition

In a parallel computer, addition is performed on complete numbers. The entire numbers, including carries, are combined in one machine cycle. Any two values, regardless of the magnitude of the numbers, can be added in the same time. Figure 3 shows parallel addition.

Numbers Being Added	00564213
Carry	1
Final Result	00565037

Figure 3. Parallel Addition

### Fixed and Variable Word Length

Fixed and variable word length describe the unit of data that can be addressed and processed by a computer system.

In fixed word length operation, information is handled and addressed in units or words containing a fixed number of positions. The size of a word is designed into the system and normally corresponds to the smallest unit of information that can be addressed for processing in the processing unit. Records, fields, characters, or factors are all expressed as words; registers, counters, accumulators, and storage are designed to accommodate a fixed word.

In variable word length operations, data handling circuitry is designed to process information serially as single characters. Records, fields, or factors may be of any practical length within the capacity of the storage unit. Information is available by character instead of by word.

Operation within a given data processing system may be entirely fixed word, entirely variable, or a combination.

In the IBM 7040 and 7044 Data Processing Systems, data are stored and processed as 36-bit words; all data manipulation operations, including arithmetic, are done in parallel. Provision is made, however, to select, shift, and perform logic operations on portions of words. Consequently, the amount of data within a word can be adjusted.

### Reading Data

All data entering the computer system must first be read by an input device and then routed to core storage. Each input device is assigned a number to serve as its address in the same way that each storage position is also assigned a location address.

A data processing procedure is normally concerned with entire files of records, which may be on magnetic tape, IBM cards, or paper tape. These files are



placed on the input device, where the computer has access to them. To read a record from a file, one or more instructions in the program activate the input device and place the record in storage.

At this point, it must be determined exactly where in storage the incoming record is to be placed, and an instruction must direct the computer to send the record to this location. Also, in the plan of manipulation, it is necessary to know at all times where to find information as needed in successive stages of processing.

These considerations involve the allocation of storage space for specific purposes in a logical and convenient manner. For example, particular fields or quantities may be used for computation. The instructions to be used later must specify the location in storage where this information from each record can be found.

The reading operation performs these distinct functions:

1. The input device is selected and made ready by the read select instruction. The device chosen is the one determined by the programmer to have access to the proper file of records. This device is selected by specifying its assigned code number (address). The read operation causes the selected input unit to transfer a record to computer storage. The record is placed in a storage area reserved for this purpose and is then available for further processing. A number of input areas may be assigned to handle several related records at a time (for example, a master record and its related transaction detail record).

2. The order of the read instructions in the program determines the sequence in which files are read. Other instructions later compare records from separate files to determine the relationship of detail to master, detail to detail, and so on.

3. The number of records to be placed in storage at one time depends on the construction of the files, the type and length of records being handled, and the available storage capacity.

### **Calculating**

Once data have been read into the computer system and placed in known locations of storage, calculation can begin. Each computer is capable of performing addition, subtraction, multiplication, and division, either as built-in operations or under program control. For most commercial applications, these operations are adequate. Even in many advanced scientific procedures, the most complex equations can be reduced to steps of elementary arithmetic. In the 7040 and 7044 systems, however, many specialized operations can be performed to make the solving of mathematical problems easier.

In every operation of simple arithmetic, at least two factors are involved: multiplier and multiplicand, divisor and dividend, and so on. These factors are operated on by the arithmetic unit of the computer to produce a result, such as a product or quotient. In every calculation, therefore, at least two storage locations are needed. One quantity is usually in core storage and the other is in the accumulator or multiplier-quotient register, which are parts of the arithmetic unit. (A register is a device with the ability to accept and hold data and to transfer the data to another register or related device.)

A calculation can be started by placing one factor in the accumulator and, at the same time, clearing this unit of any previous factors or results contained there. The address part of the instruction specifies the storage location of the first factor; the use of the accumulator or multiplier-quotient register is implied by the operation.

When one factor is properly placed in the register, the actual calculation is executed by an instruction whose operation part specifies the arithmetic operation to be performed and whose operand is the location of the second factor. The computer acts upon the two factors and produces a result, which is placed in a register. The result is returned to core storage by another instruction, which designates the storage location.

Any practical number of calculations can take place on many factors in a single series of instructions: that is, a factor may be placed in the accumulator and several other factors may be added to or subtracted from the product; division can then be executed; other operations of adding and subtracting can proceed using this quotient. Intermediate results can be stored at any time.

All calculations must take into account the algebraic sign of factors in storage or associated registers. Consequently, the computer is equipped to store and recognize the sign of a factor. With fixed word data records, the sign position automatically accompanies the word. Accumulators also include either a special sign position of storage or a sign indicator that is available to the programmer. In this way, the sign of results can be specified, together with the effect on following calculations. The computer follows the rules of algebra in all basic arithmetic operations.

The size of words, quantities, and values depends on the design of each data processing system. The exact rules governing the placement of factors, size of results, and so on vary from system to system. In all cases where a result is expected to exceed the capacity of the accumulator or storage register, the programmer must arrange (scale) his data to produce partial results and then combine these for totals. Other oper-

ations of scaling may be executed so that very large or small values and fractions may be handled conveniently.

Calculation is carried out in all computer systems at much higher speed than input or output, because reading and writing require mechanical devices and movement of documents, while calculation is performed electronically. In many commercial applications, calculation is relatively simple, and the over-all speed of the system is usually governed by the speed of the input/output units. In mathematical applications, the situation is reversed; calculation is usually complex and involved, and high calculating speeds are essential.

### Logic Operations

The sequence in which a stored program computer follows its instructions is determined in one of two ways: either it finds the instructions in consecutive storage locations or the instruction operand also designates the location of each following instruction. If instructions could be followed only sequentially in a fixed pattern, a program would follow only a single path of operation with no possibility of dealing with exceptions to the procedure and with no ability to choose alternatives based on special conditions encountered in processing data. Further, without some way of resetting the computer to repeat a given series of instructions, it would be necessary to have a complete program for each record in a file.

Consider the program illustrated in Figure 4. These instructions taken alone compute  $T$  for only one record. But by returning to the first instruction, any number of records may be processed, repeating the same program as a loop. For this purpose, another instruction is given to return to the first instruction (Figure 5).

Once this program is started, it will continue until there are no more records to process. Such program loops are common and can be terminated in many

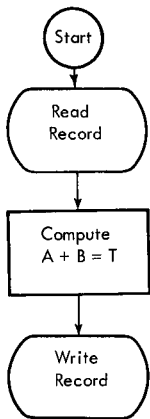


Figure 4. Block Diagram,  $A + B = T$

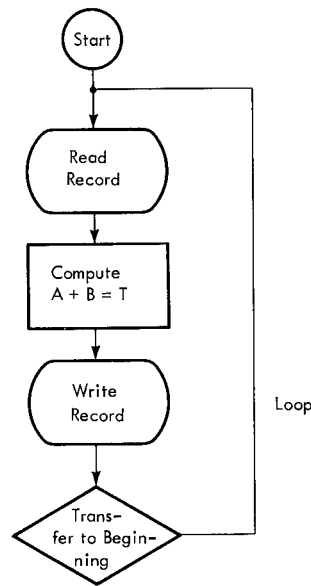


Figure 5. Program Loop

ways. For example, the computer may be instructed to examine  $T$  each time it is computed and to notify the operator when the value of  $T$  becomes negative (Figure 6).

In this case, the instruction becomes a conditional transfer. The program loop is repeated only if some predetermined condition ( $T$  is positive) is present. The computer can also be instructed to execute the program for ten records and then stop for operator intervention (Figure 7). It is assumed that the constants 10 and 1 are stored in the computer and that 1 is subtracted from 10 each time the loop is completed. After ten times around, a 0 will be in the location that contained 10 originally. A transfer or branch instruction then terminates the loop.

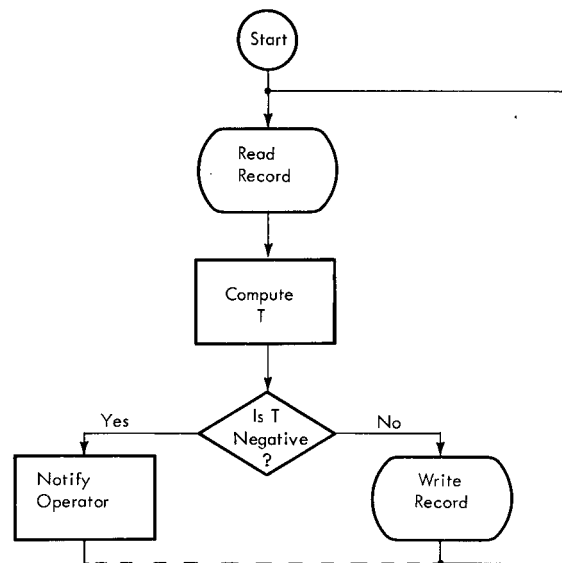


Figure 6. Conditional Transfer

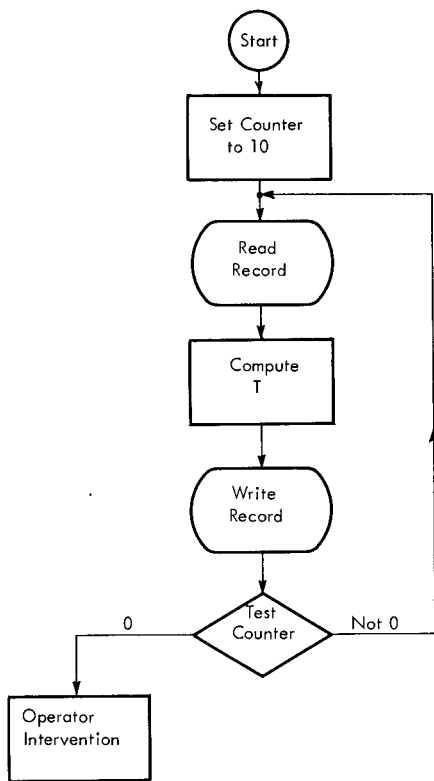


Figure 7. Program Loop under Count Control

The conditional transfer or branch operation may be used to cause a special-purpose program (subroutine) to be executed outside the normal or straight-line path of the main program. This subroutine is executed only when a predetermined exception or condition is noted by the computer.

One common example of the subroutine is checking the accuracy of records as they are read from or written on magnetic tape. As each record enters or leaves the processing unit, a read-write error indicator is tested. If the indicator has been turned on, the computer is instructed to enter a subroutine of instructions that attempts to correct the error. Figure 8 shows the program logic for such a subroutine for the reading only; a similar loop might also be included for writing.

When a reading error is detected, a transfer is made to the error subroutine. A counter is set to 10 to count the number of times a re-read will be attempted. The tape is backspaced over the error, and a second read instruction is given. Another check is made to determine if this operation is correct. If it is, a transfer returns to the main program, where computing continues.

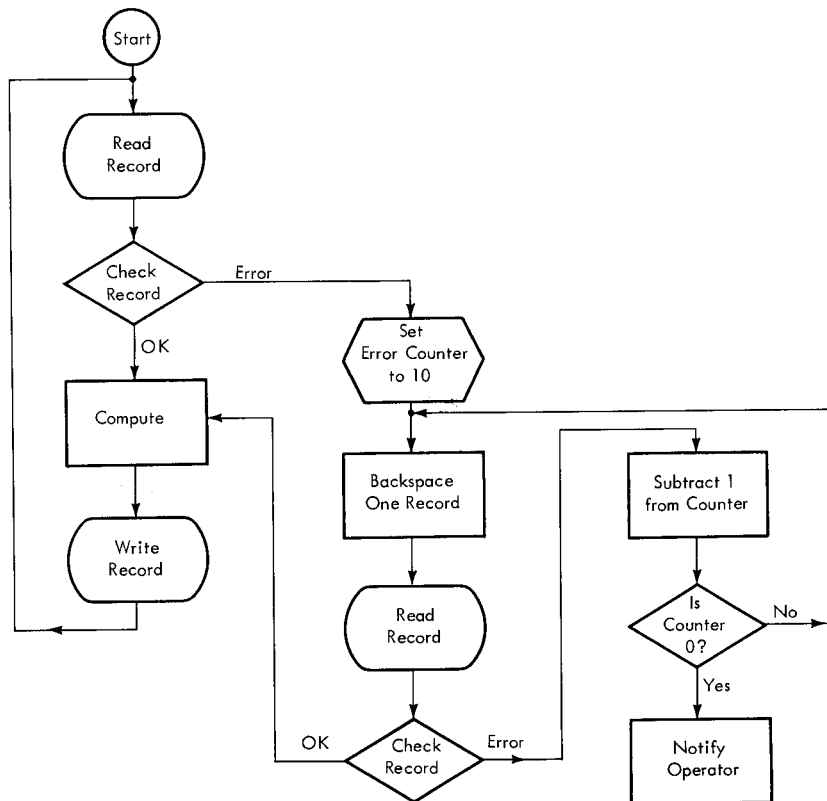


Figure 8. Tape Read Error Program Loop

If the error persists, 1 is subtracted from the counter and the counter is tested for 0. The error loop is again entered and a second re-read and check are executed. The machine can re-read ten times and, if the error is not corrected, operation is halted. Further instructions can be programmed to indicate to the operator the cause of the stop.

### Comparing

The ability of the computer to make limited decisions based on programmed logic is substantially extended by operations of comparing. Such operations enable the computer to determine if two data fields in storage are equal in value or if one is lower or higher than the other.

A value assignment for each character is built into the computer. For example, the familiar ascending sequence of the digits 0-9 assumes that the digit 9 is the highest digit of the series. In the same manner, the letter Z is assumed to be the highest letter of the alphabet. To the computer, therefore, as in any file, the number 162 is higher in sequence than 159, and the name Jones is lower than the name Smith. Special characters, such as /, @, \*, or - may also be included because these characters must be manipulated as data for report printing and other special purposes.

Comparing operations are used to program the sequence checking of files, sorting procedures, or the rearrangement of records in some desired order. The comparison of an identifying field in one record with that of another insures that the proper records are processed. Out-of-sequence records are detected by the comparison operation.

The two fields to be compared are placed in core storage. One field is then placed in an accumulator register, and a compare instruction is given to compare this field against the second specified field. The results of comparison are registered in high, low, or equal indicators, which are then interrogated to determine their condition. If a particular indicator is on, an automatic operation transfers the program to a subroutine that continues processing according to the result of the comparison.

Figure 9 shows a typical program arrangement for sequence checking a single file of records. All records in the file are assumed to be in ascending sequence by account number. An input area - where records are received, one at a time, from an input unit - is set aside in storage. A second area is also reserved in storage to store the account number from the preceding record. The purpose of this area is to allow comparison of the account number of the incoming record with the corresponding field of the previous record.

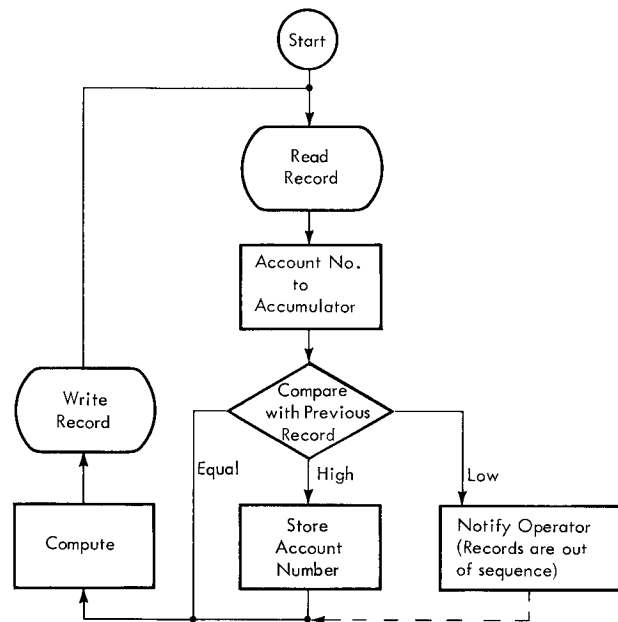


Figure 9. Sequence Checking

If the file is in ascending sequence, the incoming record should always be higher than the record that preceded it. When duplicate records are encountered, the incoming record is equal to the preceding one. If any incoming record is lower than the previous record, it is recognized as an out-of-sequence condition, and an error is signaled to the operator. The out-of-sequence record may be noted, and corrective action may be either taken by the operator or programmed as a subroutine. After each high comparison, the account number field is placed in storage where it may be compared with the next record.

### Instruction Modification

Some of the preceding examples have shown how branching or transfer instructions can cause the computer to follow a varied path through the program. The routine to be executed depends on the result of a previous comparison or a test of indicators that have been set by a zero in a counter, an error condition, and so on.

Another method of varying the program is by changing or modifying the operation part of the instructions themselves. Instruction modification, for example, can be used to set up a program switch, which can cause the machine to take one of two alternate paths. The switch (which is an electronic switch) is turned on or off by instruction. Figure 10 shows the use of the switch.

Assume that two files are being read. The files are in sequence by a common identifying field, such as part number, account number, or employee number. One file is a master file; the second is a transaction file

that represents adjustments to the master. Three conditions may be encountered in applying the transactions to the corresponding master files:

1. One or more transactions may match a single master record.
2. There may be no transactions for a master record.
3. There may be transactions that do not match a master — these are errors.

It is necessary to process the two files in step; that is, each transaction record must be compared against a corresponding master record, if there is one. If several transactions apply to the same master record, the transaction file must continue reading without reading a new master record. Conversely, if a master record is read in without a corresponding transaction, this record is written out unchanged and the following master is read in. The reading and writing of master records continue until a matching transaction is found.

Figure 10 shows that one master record is read in first. A switch instruction is inserted between the reading of the master and the transaction. As operations

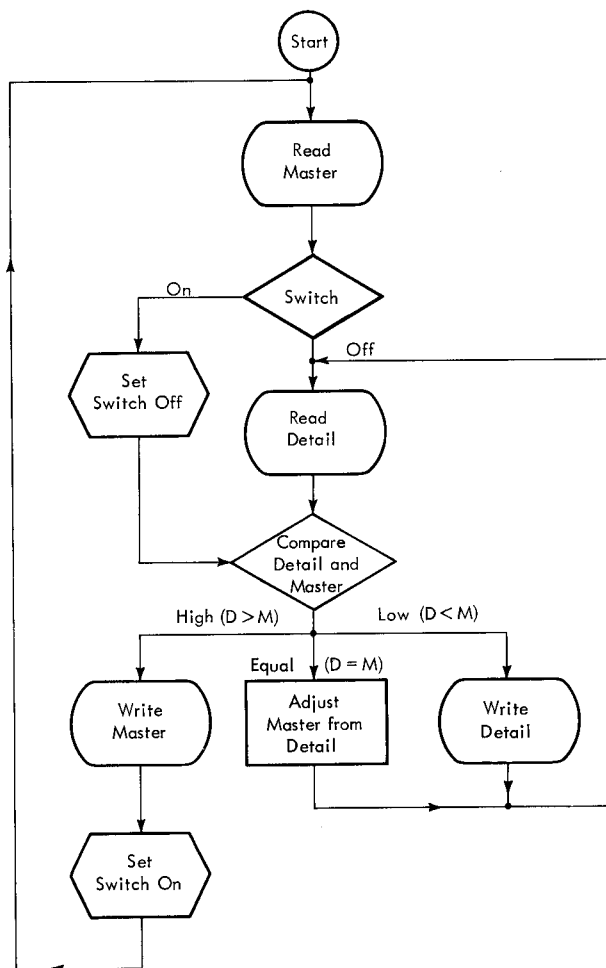


Figure 10. Program Switch

begin, this switch is turned off, allowing one transaction to be read in. The identifying field of the transaction is compared against the master. If they are equal, the master is adjusted and a second transaction is read in. If this transaction is not to be applied against the master (which is still in storage), it should be high when compared. The previously adjusted master is then written out and the switch is turned on. A new master is then placed in storage but, because the switch is on, a transaction is not read; instead the machine transfers directly to the compare instruction. The switch is turned off each time this happens. Operation continues, with comparison for each new record placed in storage. If a transaction is low (has no master record), it is written out on a separate output unit, and a new transaction is then read in.

The switch, when on, acts as an instruction with an operation part specifying an unconditional transfer. The address part is the location of the compare instruction. To turn the switch off, the operation part is changed to no operation by the program. In this case, the computer ignores the instruction and proceeds to the following instruction: read a transaction.

#### Address Modification

Because the address portion of instructions may be treated as data, instruction addresses can be modified by arithmetic, reducing the number of instructions in a program and conserving storage capacity for data or other factors. One instruction, or a single series of instructions, can serve to address variable locations in storage.

For example, the address part of instructions that select the devices of a system may be modified by other instructions in the program. One use of this type of modification is the selection of alternate magnetic tape units when an end-of-file or end-of-reel condition is signaled. A reading or writing operation may then continue without interruption on an alternate unit while the first unit is rewinding or standing by for reel change. When a tape file is made up of more than one reel, reading or writing may proceed from reel to reel with minimum lost time.

Assume that the addresses of two tape units are 0201 and 0203. The sum of the units and tens positions of these addresses is stored as a constant factor 04. When an end-of-file condition is signaled by tape unit 0201, a transfer is made to a subroutine. In the subroutine, the units and tens positions of the tape unit being used (01) are placed in an accumulator. The constant 04 is subtracted to obtain minus 03 as a result. This result is then used as the tape unit address, converting it from 0201 to 0203. The sign of the result is ignored.

The subroutine then transfers back to the main routine and uses tape unit 0203. When end-of-file is signaled from this unit, the constant 04 is subtracted from 03 to obtain the result minus 01. Using this result changes the tape address from 0203 to 0201. The address of the select instruction alternates between 0201 and 0203 each time an end-of-file is signaled.

### Indexing

In the 7040 and 7044 computers, the address portion of an instruction can be modified by adding or subtracting variable quantities contained in one or more special-purpose registers called index registers.

Computers with an indexing feature use an instruction format that allows a particular register or word to be specified as a part of the instruction.

Assume that fifty quantities are placed in ascending word positions of storage from locations 1001 to 1050 inclusive and that these quantities are to be added to the contents of an accumulator. Without indexing or address modification, it is necessary to repeat an add instruction fifty times with the address of each instruction incremented by 1. For example: ADD 1001, ADD 1002, ADD 1003, and so on.

With indexing, the add instruction can be written as ADD 1051 with the address decremented by an index register containing the quantity 50. The address in storage remains 1051, but the computer calculates and uses an effective address of 1051 minus 50, or 1001. When the add instruction is executed, the contents of the index register are also decremented by 1 (leaving a remainder of 49) and are tested for 0. When the same add instruction is re-executed and is again decremented by the contents of the same index register, the effective address is 1051 minus 49, or 1002. Each time the index register is decremented, it is also tested for 0.

If a program loop is formed to repeat this process, the effective address of the add instruction is stepped up 1 each time it is executed (as the index register contents are stepped down). When the index register equals zero, all 50 quantities will have been added and the loop is terminated. The computer has consequently performed 50 operations using the same add instruction. Figure 11 is a flow diagram of the index loop.

The first instruction places the quantity 50 in index register 4. An add instruction, with an address 1051, also specifies as part of its operand that the given address is to be modified by the quantity contained in index register 4. The next instruction is transfer on index, which means: reduce the contents of index register by 1; if the contents of the register are greater than 0, transfer to repeat the add instruction; if the contents

of the index register equal 0, continue to the next instruction in the program.

The indexing feature greatly simplifies programming of repetitious calculations or other operations and reduces the number of instructions required.

### Indirect Addresses

All instruction addresses discussed in preceding illustrations are classified as direct, that is, they refer directly to the location of data or other instructions in storage, they select a system component, or they specify the type of control to be exercised.

Addresses may also be indirect. Such an address can refer only to a storage location that contains another address. The second address in turn refers to the location of data, a system component, or a control function.

Indirect addressing is particularly useful in performing address modification. For example, in a program it may be necessary to refer a number of instructions to a value which changes with each program iteration. Without indirect addressing, a number of modification instructions would be needed.

However, if the instructions are indirectly addressed to one core storage location, that location can contain a single address, the address of the values being used by the program. Therefore, to change or modify all instruction addresses, it is only necessary to modify the single effective address to which the instructions refer (Figure 12). In this text, the asterisk (\*) is used with the operation code to designate indirect addressing. Any number of indirect addresses throughout a program may refer to a single effective address. In Figure 12, each indirectly addressed clear and add instruction (CLA\* 4069) would bring in the contents of core location 2000 instead of location 4069.

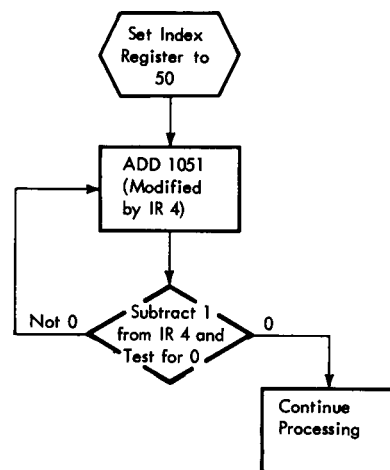


Figure 11. Indexing Loop

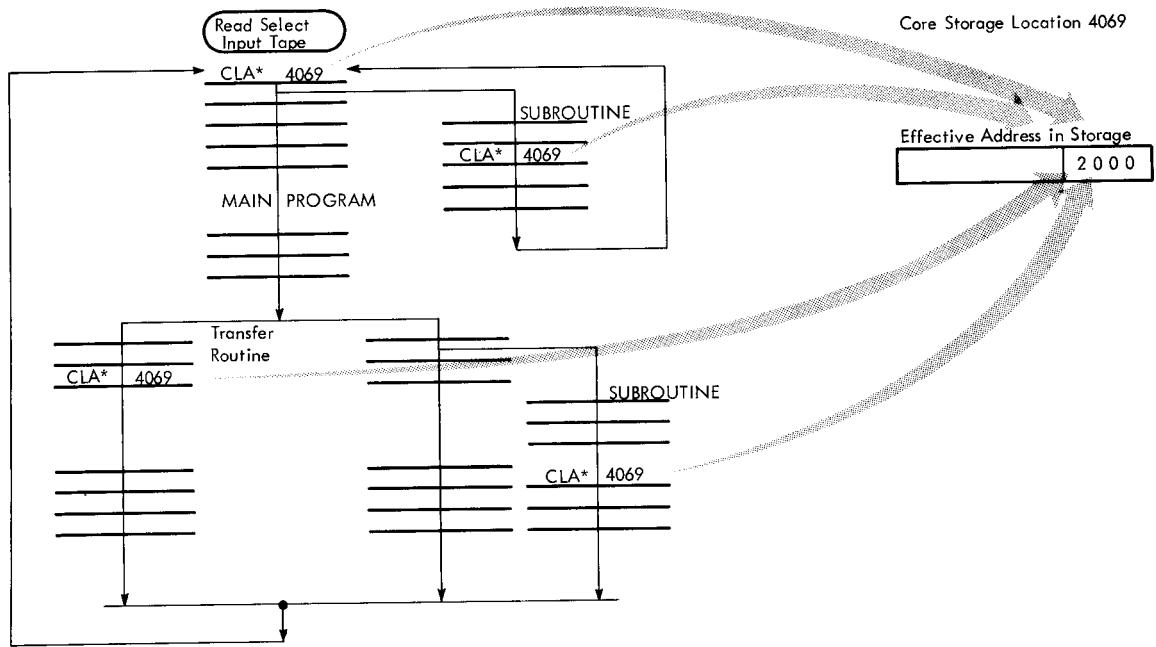


Figure 12. Indirect Address

# Computer Data and Instructions

## Numbers Concept

The common decimal system, with its ten different symbols, is learned by most people early in their training. This system serves very well for counting. Why then, should computers, which are designed to assist engineers, businessmen, and scientists, be designed to use a different system of notation?

The decimal system is built around the base ten and uses the 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 symbols. Combining these symbols and a place system for their arrangement, any number can be expressed, no matter how large or how small. The value of each symbol depends on its place in a row of symbols. For example, the symbol 1, by itself, has a place value of 1. Combined with another symbol, as in 21, the 1 symbol still has a place value of 1. Reverse the symbols, however, (12) and the 1 symbol now has a place value of 10.

This concept can be readily applied to any other number system. For example, imagine a number system containing only the symbols 0, 1, 2, 3, and 4. Since there are five symbols used, the system is called quinary or, more commonly, a base 5 system. To count in this system, the first symbol used is the 0. This is followed by the 1, 2, 3, and 4. At this point, all five symbols have been used. The next step is to assign the decimal value of five to the 1 symbol by placing it one position to the left and combining it with the 0 symbol (10). This combination is then followed by the 11, 12, 13, and 14 combinations. The third symbol in the system (2) is then assigned the decimal value of ten and is combined with the 0 giving the combination 20. This is followed by 21, 22, 23, 24, 30, and so forth.

The following table shows the arrangement of symbols used to represent the same values in each system of notation.

DECIMAL	QUINARY	DECIMAL	QUINARY
0	0	10	20
1	1	11	21
2	2	12	22
3	3	13	23
4	4	14	24
5	10	15	30
6	11	16	31
7	12	17	32
8	13	18	33
9	14	19	34

The main difficulty in using an unfamiliar number system is recognizing the new values assigned to familiar symbols. For example, to add the decimal

symbols 3 and 4 and get a decimal result of 7 is simple for anyone acquainted with the decimal system. To add the quinary symbols 3 and 4 and get a quinary result of 12 is more difficult because of limited use of the quinary system.

## Arithmetic Tables

The construction of arithmetic tables makes operations faster and easier. Figure 13 shows sample add tables for both decimal and quinary systems.

Decimal						Quinary							
	0	1	2	3	4	5		0	1	2	3	4	10
0	0	1	2	3	4	5	0	0	1	2	3	4	10
1	1	2	3	4	5	6	1	1	2	3	4	10	11
2	2	3	4	5	6	7	2	2	3	4	10	11	12
3	3	4	5	6	7	8	3	3	4	10	11	12	13
4	4	5	6	7	8	9	4	4	10	11	12	13	14
5	5	6	7	8	9	10	10	10	11	12	13	14	20
6	6	7	8	9	10	11	11	11	12	13	14	20	21

Figure 13. Add Tables

To use these tables, the symbols being added (3 and 4 in the decimal table) are located, one on the top and the other on the left side of the table. Lines are then projected until they meet. The value at the intersection is the result of the addition. Using the quinary table:  $4 + 3 = 12$ ,  $11 + 4 = 20$ ,  $2 + 4 = 11$ , and so forth. The results are expressed in quinary values.

The same principle may be applied to other arithmetic processes. Multiply tables for both systems are shown in Figure 14. The use of these tables is the same as with the add tables; only the results differ. For example,  $3 \times 4$  with the decimal table gives the result of 12, while  $3 \times 4$  with the base 5 table gives the result 22; both results represent the same quantity.

Decimal						Quinary					
	0	1	2	3	4		0	1	2	3	4
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	1	0	1	2	3	4
2	0	2	4	6	8	2	0	2	4	11	13
3	0	3	6	9	12	3	0	3	11	14	22
4	0	4	8	12	16	4	0	4	13	22	31

Figure 14. Multiply Tables



## Binary Mode

Computers function in what is called a binary mode. This term simply means that the computer components can indicate only two possible states or conditions. Therefore, the binary mode system may also be called a base 2 system. For example, the ordinary light bulb operates in a binary mode; it is either on, producing light; or it is off, not producing light. The presence or absence of light indicates whether the bulb is on or off. Likewise, within the computer, transistors are either conducting or not conducting, magnetic materials are magnetized in one direction or in the opposite direction; and specific voltage potentials are present or absent (Figure 15). The binary modes of operation of the components are signals to the computer, as the presence or absence of light from an electric light is to a person.

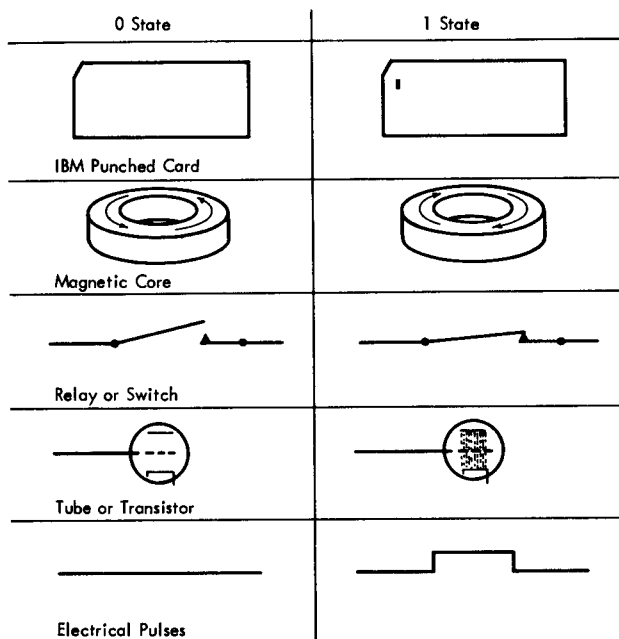


Figure 15. Binary Indicators

Representing data within the computer is accomplished by assigning or associating a specific value to a binary indication or group of binary indications. For example, a device to represent values could be designed with four electric light bulbs and switches to turn each bulb on or off (Figure 16).

The bulbs are assigned arbitrary values of 1, 2, 4, and 8. When a light is on, it represents the value associated with it. When a light is off, the value is not

considered. With such an arrangement, the single value represented by the four bulbs will be the numeric sum indicated by the lighted bulbs.

Values 0 through 15 can be represented. The value 0 is represented by all lights off; the value 15, by all lights on; 9, by having the 8 and 1 lights on and the 4 and 2 lights off; 5, with the 1 and 4 lights on and the 8 and 2 lights off; and so on.

The value assigned to each bulb or indicator in the example could have been something other than the values used. This change would involve assigning new values and determining a scheme of operation. In a computer, the values assigned to a specific number of binary indications become the code or language for representing data.

Because binary indications represent data within a computer, a binary method of notation is used to illustrate these indications. The binary system of notation uses only two symbols, zero (0) or one (1), to represent all quantities. In any single position of binary notation, the 0 represents the absence of a related or assigned value and the 1 represents the presence of a related or assigned value. Using the light bulbs in Figure 16, for example, the binary notation 0101 would represent a decimal 5.

The binary notations 0 and 1 are commonly called bits. The 0 bit is described as no bit and the 1 bit is described as a bit. Although 0 or 1 bits are necessary to illustrate the condition of a binary indication or a group of binary indications, the 1 bits are the bits generally referred to. For example, the binary notation 0101 of Figure 16 would be described as having a bit in the 1 and 4 bit positions. The assumption is that there are no bits (0 bits) in the 2 and 8 bit positions.

## Binary Number System

In some computers, the values associated with the binary notation are related directly to the binary number system. This system is not used in all computers,

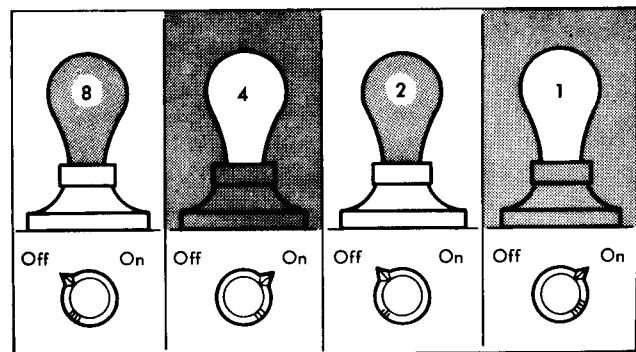


Figure 16. Representing Decimal Data

but the method of representing values using this numbering system is useful in learning the general concept of data representation.

The common decimal number system uses ten symbols or digits to represent all quantities, and the place value of the digits signifies units, tens, hundreds, thousands, and so on. The binary or base 2 number system uses only two symbols or digits: 0 and 1. The position value of the bit symbols (0 or 1) is based on the progression of powers of 2; the units position of a binary number has the value of 1; the next position, a value of 2; the next, 4; the next, 8; the next, 16; and so on (Figure 17).

8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
------	------	------	------	-----	-----	-----	----	----	----	---	---	---	---

Figure 17. Place Value of Binary Numbers

In pure binary notation, the binary digits or bits indicate whether the corresponding power of 2 is absent or present in each position of the number. The 1 bit represents the presence of the value and the 0 bit represents the absence of the value. The place value of the digits does not signify units, tens, hundreds, or thousands, as in the decimal system; instead, the place value signifies units, twos, fours, eights, sixteens, and so on. Using this system, the quantity 12, for example, is expressed with the symbols 1100, meaning  $(1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (0 \times 2^0)$  or  $(1 \times 8) + (1 \times 4) + (0 \times 2) + (0 \times 1)$ .

Figure 18 shows the binary representation of the decimal values 0 through 9. Note that the decimal digits 0 through 9 are expressed by four binary digits. The system of coding or expressing decimal digits in an equivalent binary value is called binary coded decimal (BCD). For example, the decimal digits 2, 6, 5, 4, 9, and 8 would appear in binary coded decimal form as shown in Figure 19.

Decimal Value	Place Value			
	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Figure 18. Binary Representations

Although binary numbers, in general, have more terms than their decimal counterparts (about 3.3 times as many), computation in the binary system is quite simple.

For addition, it is only necessary to remember three rules:

1. Zero plus zero equals zero.
2. Zero plus one equals one.
3. One plus one equals zero with a carry of one to the next position on the left.

To see how the rules work, consider the addition of  $15 + 7$ , with these numbers expressed in binary notation:

	SIXTEENS	EIGHTS	FOURS	TWOS	ONES
(Carries)	(1)	(1)	(1)	(1)	
	0	1	1	1	1 = 15
	0	0	1	1	1 = 7
	1	0	1	1	0 = 22

In the ones column, we have  $1 + 1$  for a sum of 0 and a 1 carried to the twos column. In the twos column, we have  $1 + 1$  for a sum of 0, but we must also add the carry from the ones column, making a final sum of 1 with a carry to the fours column. In the eights column, we have a  $1 + 0$  giving a sum of 1, but adding in the carry from the fours column makes the final sum 0 with a carry to the sixteens column. In this column, we have  $0 + 0$ , giving a sum of 0 and to this we add the carry from the eights column, making a final sum of 1.

The resultant sum of the addition contains 1's in the sixteens, fours, and twos column, which is the binary representation of 22, the correct sum of 15 plus 7 ( $16 + 4 + 2 = 22$ ).

The rules for subtraction of binary digits are equally simple:

1. Zero minus zero equals zero.
2. One minus one equals zero.
3. One minus zero equals one.
4. Zero minus one equals one, with one borrowed from the left.

Decimal Digits	2	6	5	4	9	8
Binary Value	0010	0110	0101	0100	1001	1000
Place Value	8421	8421	8421	8421	8421	8421

Figure 19. Binary Coded Decimal

Using the same numbers as we did in the addition, the subtraction is:

	SIXTEENS	EIGHTS	FOURS	TWOS	ONES
(Borrows)	(0)	(0)	(0)	(0)	(0)
	0	1	1	1	1 = 15
-	0	0	1	1	1 = 7
	0	1	0	0	0 = 8

In the ones column we have 1 - 1 for a sum of 0 with no borrows. The same procedure occurs in the twos and fours columns. In the eights column, we have 1 - 0 for a sum of 1. In the sixteens column, we have 0 - 0 for a sum of 0. With the subtraction finished, we have 1's in the eights column only, signifying the answer to be 8.

For multiplication, only three rules are needed:

1. Zero times zero equals zero.
2. Zero times one equals zero; no carries are considered.
3. One times one equals one.

In the binary multiplication table, all that is necessary when multiplying one number (multiplicand) by another (multiplier) is to examine the multiplier digits one at a time and, each time a 1 is found, add the multiplicand into the result, and each time a 0 is found add nothing. The multiplicand must be shifted for each multiplier digit, but this is no different from the shifting done in the decimal system.

An example of binary multiplication is  $26 \times 19$ :

DECIMAL	BINARY
26 = 16 + 8 + 0 + 2 + 0	= 11010
× 19 = 16 + 0 + 0 + 2 + 1	= 10011
Using the rules, the product is arrived at by a series of adding the multiplicand and shifting whenever a 1 is in the multiplier.	11010 11010 00000 00000 11010 <hr style="width: 50px; margin-left: 0;"/> 111101110

Interpreting the binary result of the multiplication by using the ones, twos, fours, . . . etc. system, we find:

$$256 + 128 + 64 + 32 + 0 + 8 + 4 + 2 + 0$$

which equals 494, proving the problem.

Binary division is accomplished by applying similar concepts. From the examples of addition, subtraction, and multiplication, you can see that whatever operation the computer is working on is accomplished by repetitive addition.

The computer operates internally using the binary system. However, it is able to convert from one system to another by use of a stored program. Thus,

input/output data may be expressed in decimal (or any other) form when the programmer finds it convenient to do so.

### Octal Number System

It has been noted that binary numbers require about three times as many positions as decimal numbers to express the equivalent number. This is not much of a problem to the computer; however, in talking and writing or in communicating with the computer, these binary numbers are bulky. A long string of 1's and 0's cannot be effectively transmitted from one individual to another. Some shorthand method is necessary.

The octal number system fills this need. Because of the simple relationship of octal to binary, numbers can be converted from one system to another by inspection. The base or radix of the octal system is 8. This means there are eight symbols: 0, 1, 2, 3, 4, 5, 6, and 7. There are no 8's or 9's in this number system. The important relationship to remember is that three binary positions are equivalent to one octal position. The following table is used constantly when working on or about the computer.

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

At this point, all eight symbols have been used, and a carry to the next higher position of the number is necessary.

BINARY	OCTAL
001 000	10
001 001	11
001 010	12
001 011	13
001 100	14

and so on.

Remember that as far as the internal circuitry of the computer is concerned, it only understands binary. But an operator can look at a series of lights on the computer console showing binary 1's and 0's, for example:

100 011 101 000 111 010 100 011 110 111 101 001

and say that the lights represent the octal value 435072436751. This is easier to state than the actual binary 1's and 0's.

### Number Conversions

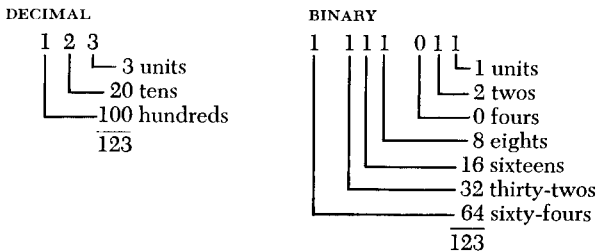
Before converting numbers from one system to another, it is best to review what a number represents. In the decimal system, a number is represented or expressed by a sum of terms. Each individual term consists of a product of a power of ten and some integer from 0 to 9. For example, the number 123 means 100 plus 20 plus 3. This may also be expressed as:

$$(1 \times 10^2) + (2 \times 10^1) + (3 \times 10^0)$$

Ten is said to be the base or radix of this system. Radix is defined as an integer used in a system of notation whereby all numbers are expressed as powers of the integer. In the decimal system, the radix is 10; in the binary system, it is 2. If 2 is chosen as the base, numbers are said to be represented in the binary system. Consider the binary number 1 111 011. What do these zeros and ones represent? They represent the coefficients of the ascending powers of 2. Expressed in another way the number is:

$$(1 \times 2^6) + (1 \times 2^5) + (1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0)$$

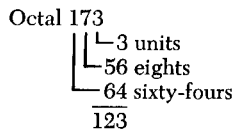
The places do not have the meaning of units, tens, hundreds, thousands, etc., as in the decimal system; instead they signify units, twos, fours, eights, sixteens, etc. In applying the above information, the decimal number 123 breaks down in both systems as:



In the octal system, a number is represented in the same manner, except that the base is 8. The digits of the number represent the coefficients of the ascending powers of 8. Consider the octal number:

$$\begin{aligned} 173 &= (1 \times 8^2) + (7 \times 8^1) + (3 \times 8^0) \\ &= 64 + 56 + 3 \\ &= 123 \text{ (decimal)} \end{aligned}$$

Similarly:



By remembering what a number represents in the binary or octal system, you can convert the number

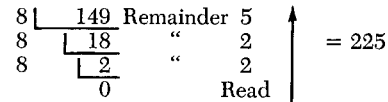
to its decimal equivalent by the method shown. As the numbers get bigger, this method becomes quite impractical. The following section provides detailed methods for converting from one system to another.

### Integers

#### DECIMAL TO OCTAL

Convert decimal number 149 to its octal equivalent.

*Rule:* Divide the decimal number by 8 and develop the octal number:

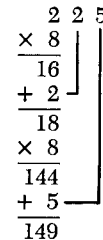


The original number to be converted is divided by 8. The remainder of this first division becomes the low-order digit of the conversion (5). The quotient (received from the first division) is then divided by 8. Again the remainder becomes a part of the answer (next higher order, 2). This method is continued until the quotient is smaller than the divisor. The final quotient is considered the high order of the conversion (2).

#### OCTAL TO DECIMAL

Convert octal number 225 to its decimal equivalent.

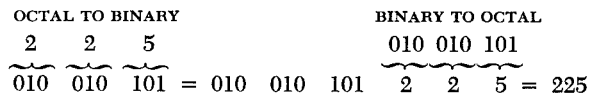
*Rule:* Multiply by 8 and add, as in the example:



The high-order digit is multiplied by 8 and the next lower-order digit is added to the result. The resultant answer is then multiplied by 8 and the next lower-order digit is added to the result. When the low-order digit has been added to the answer, the process ends. In the following examples, where multiplication or division is used, detailed explanations are not given because the operations are similar.

#### OCTAL TO BINARY AND BINARY TO OCTAL

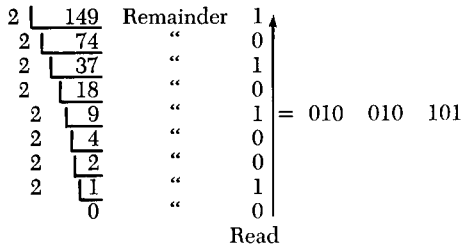
*Rule:* Express the number in binary groups of three:



**DECIMAL TO BINARY**

Convert 149 to its binary equivalent.

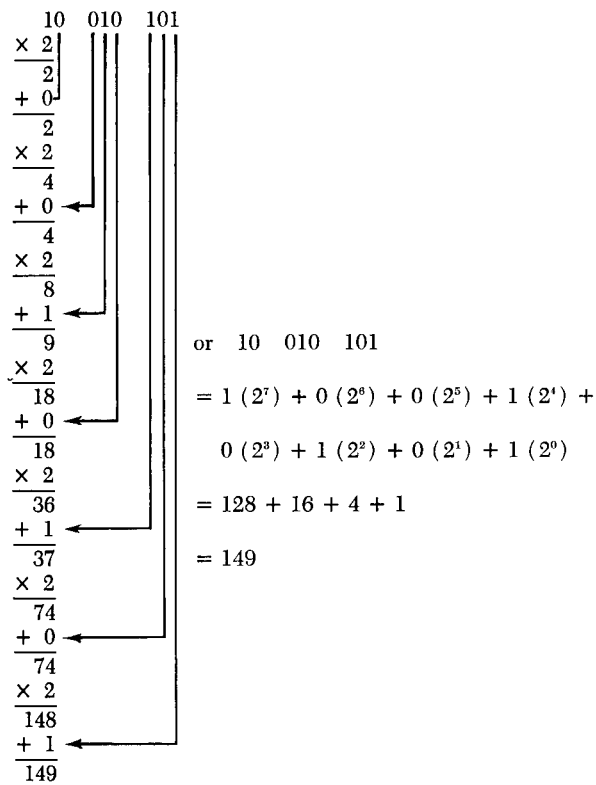
*Rule:* Divide the decimal number by 2 and develop as in the example:



**BINARY TO DECIMAL**

Convert 010 010 101 to its decimal equivalent.

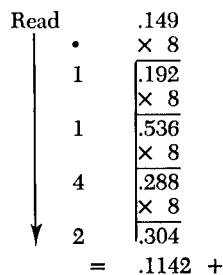
*Rule:* Multiply by 2 and add as in the example:



**Fractions**

**DECIMAL TO OCTAL**

*Rule:* Multiply by 8 and develop the octal number as in the example:



**OCTAL TO DECIMAL**

*Rule:* Express as powers of 8, add and divide as in the example:

$$\begin{aligned}
 .1142 &= 1 (8^{-1}) + 1 (8^{-2}) + 4 (8^{-3}) + 2 (8^{-4}) \\
 &= 1/8 + 1/64 + 4/512 + 2/4096 \\
 &= 610/4096 \\
 &= .1489 \text{ plus} \\
 &\text{or } .149
 \end{aligned}$$

**OCTAL TO BINARY AND BINARY TO OCTAL**

*Rule:* The same rule applies for fractions as for whole numbers:

$$\begin{array}{cccc}
 \underbrace{.1}_{.001} & \underbrace{1}_{001} & \underbrace{4}_{100} & \underbrace{2}_{010} \\
 \underbrace{.001}_{.1} & \underbrace{001}_1 & \underbrace{100}_4 & \underbrace{010}_2
 \end{array}$$

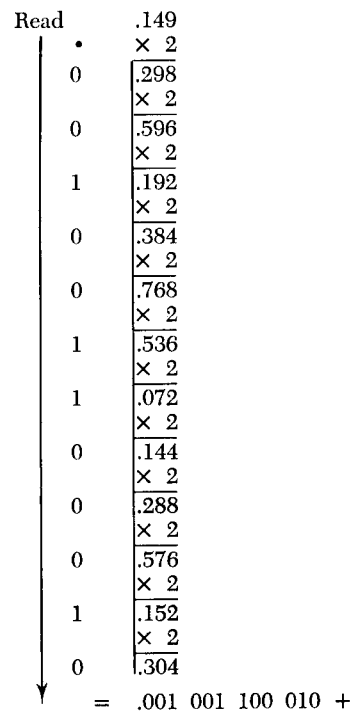
**BINARY TO DECIMAL**

The same rule applies as for whole numbers:

$$\begin{aligned}
 .001 \ 001 \ 100 \ 010 \\
 &= 1 (2^{-3}) + 1 (2^{-6}) + 1 (2^{-7}) + 1 (2^{-11}) \\
 &= 1/8 + 1/64 + 1/128 + 1/2048 \\
 &= 305/2048 \\
 &= .1489 \text{ plus} \\
 &\text{or } .149
 \end{aligned}$$

**DECIMAL TO BINARY**

The same rule applies as for whole numbers:



### Improper Fractions

#### DECIMAL TO BINARY

Convert 149.149 to its binary equivalent. This requires conversion from decimal to octal and then to binary:

$$\begin{array}{r}
 8 \overline{) 149} \quad \text{remainder } 5 \\
 \underline{8 \phantom{0} 8} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 18 \phantom{0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 \underline{8 \phantom{0} 0} \phantom{0} \phantom{0} \phantom{0} \phantom{0} \\
 0
 \end{array}
 \quad
 \begin{array}{c}
 5 \\
 2 \\
 2 \\
 \uparrow \\
 \text{Read}
 \end{array}
 \quad
 \begin{array}{c}
 .149 \\
 \times 8 \\
 \hline
 .192 \\
 \times 8 \\
 \hline
 .536 \\
 \times 8 \\
 \hline
 .288 \\
 \times 8 \\
 \hline
 .304
 \end{array}$$

$$\begin{array}{c}
 1 \\
 1 \\
 4 \\
 2 \\
 \downarrow \\
 \text{Read}
 \end{array}$$

$$= \underbrace{2}_{010} \underbrace{2}_{010} \underbrace{5}_{101} \cdot \underbrace{1}_{001} \underbrace{1}_{001} \underbrace{4}_{100} \underbrace{2}_{010}$$

$$149.149_{10} = 225.1142_8 = 010\ 010\ 101.001\ 001\ 100\ 010_2$$

#### BINARY TO DECIMAL

This requires conversion from binary to octal and then to decimal:

$$\begin{array}{r}
 \begin{array}{c}
 010 \quad 010 \quad 101 \cdot 001 \quad 001 \quad 100 \quad 010 \\
 \hline
 2 \quad 2 \quad 5 \cdot 1 \quad 1 \quad 4 \quad 2 \\
 \hline
 \times 8 \\
 16 \\
 + 2 \\
 \hline
 18 \\
 \times 8 \\
 144 \\
 + 5 \\
 \hline
 149
 \end{array}
 \end{array}$$

$$\begin{array}{c}
 \frac{1}{8} + \frac{1}{64} + \frac{4}{512} + \frac{2}{4096} = \\
 \frac{610}{4096} = .149
 \end{array}$$

As with decimal-to-binary, conversion of the integer and fraction parts is performed independently.

### Computer Codes

The method or system used to represent (symbolize) data is a code. In the computer, the code relates data to a fixed number of binary indications (symbols). For example, a code used to represent numeric and alphabetic characters may use seven positions of binary indication; by the proper arrangement of the binary indications (bit, no bit), all characters can be represented by a different combination of bits.

Some computer codes in use are: seven-bit alphameric code, two-of-five fixed count code, bi-quinary code, six-bit numeric code, and the binary system.

#### Code Checking

Most computer codes are self-checking; that is, they are provided with a built-in method of checking the validity of the coded information. This code checking

occurs automatically within the system as the data processing operations are carried out. The method of validity checking is a part of the design of the code.

In some codes, each unit or character of data is represented by a specific number of bit positions, and these must always contain an even number of 1 bits. Different characters are made up of different combinations of 1 bits, but the number of 1 bits in any valid character is always even. With this code system, a character with an odd number of 1 bits is detected and an error is indicated. Likewise, a code may be used in which all characters must have an odd number of 1 bits; an error is indicated when a character with an even number of 1 bits is detected.

This type of checking is known as a parity check. Codes that use an even number of 1 bits are said to have even parity. Codes that use an odd number of bits are said to have odd parity.

In other codes, the number of 1 bits present in each unit of data is fixed. For example, a code may use five bit positions to code all digits but only two 1 bits will be present in each digit. Digits having more or fewer than two 1 bits cause an error indication. This system of checking is known as a fixed count check.

#### Seven-Bit Alphameric Code (Binary Coded Decimal)

In this code, all characters—numeric, alphabetic, and special—are represented (coded) using seven positions of binary notation. These positions are divided into three groups: one check position, two zone positions, and four numeric positions (Figure 20).

Check Bit	Zone Bits	Numeric Bits
C	B A	8 4 2 1

Figure 20. Bit Positions, Seven-Bit Alphameric Code

The four numeric positions are assigned decimal values of 8, 4, 2, and 1 and represent, in binary coded decimal form, the numeric digits 0 through 9 (Figure 21). Note that 0 is represented as 1010, actually the binary number for 10. The B and A zone bits are not present (they are 00) when the numeric digits 0 through 9 are represented.

Combinations of zone and numeric bits represent alphabetic and special characters. The B and A bits provide three bit combinations: 10, 01, and 11. Figure 22 shows the zone and numeric bit combinations used

Decimal Value	Place Value			
	8	4	2	1
0	1	0	1	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Figure 21. Seven-Bit Alphameric Code (Numeric Part)

to represent numeric, alphabetic, and special characters in the IBM 705 and 7080 Data Processing Systems and on IBM magnetic tape. In other systems using this code, there may be special characters not shown; however, these characters follow the same scheme of bit arrangement.

The C position, known as the check bit, is used for code checking only. Because the seven-bit alphameric code is an even parity code, the number of bits that represent a character must have an even number of bits, or the character is considered invalid. The check bit is present in a character when the sum of the zone and numeric bits representing the character is odd. If the number of bits in a character is even without the C bit, the C bit is not used.

### Problems

1. Convert  $89_{10}$  to its octal equivalent.
2. Convert  $010001110010111_2$  to its decimal equivalent.
3. Convert the fraction  $.358_{10}$  to its octal equivalent.
4. Convert the improper fraction  $139.247_{10}$  to its binary equivalent.

With the next four problems, it is necessary to first convert the decimal numbers and then perform the arithmetic operation.

5. Add  $18_{10}$  and  $92_{10}$  in binary.
6. Subtract  $34_{10}$  from  $71_{10}$  in binary.
7. Multiply  $17_{10}$  times  $43_{10}$  in binary.
8. Divide  $448_{10}$  by  $14_{10}$  in binary.
9. Make a binary add and a binary multiply table.

### Code Definitions

Four code structures are used with the 7040 and 7044 systems and input-output equipment. Each code is a specific system of representing numeric, alphabetic, and special characters. Figure 23 shows the codes and relations between them.

Two sets of graphics are used. One is designed for report writing and the other for programming language. A print head for the console typewriter is available for each graphic set. For other IBM Printers, special character arrangement A agrees with the report writing set and special character arrangement H agrees with the programming language set.

Code headings in Figure 23 are:

*H* (standard IBM card code) defines the combination of punches used to represent each of the 64 code combinations.

9 (as used in IBM 704, 709, 7090, 7094) shows the same characters as they are normally placed in internal storage of the 7040 and 7044 systems. The six-bit code groups are represented as two octal digits.

5 (as used in the 705, 7080, and on binary coded decimal (BCD) magnetic tape) shows the same characters as they normally appear on BCD magnetic tape for communication with other IBM magnetic tape equipment. Note that this representation permits only 63 code combinations, not 64.

14 (as used in the 1401, 1410, and 1414) shows the same characters as they normally appear in the internal

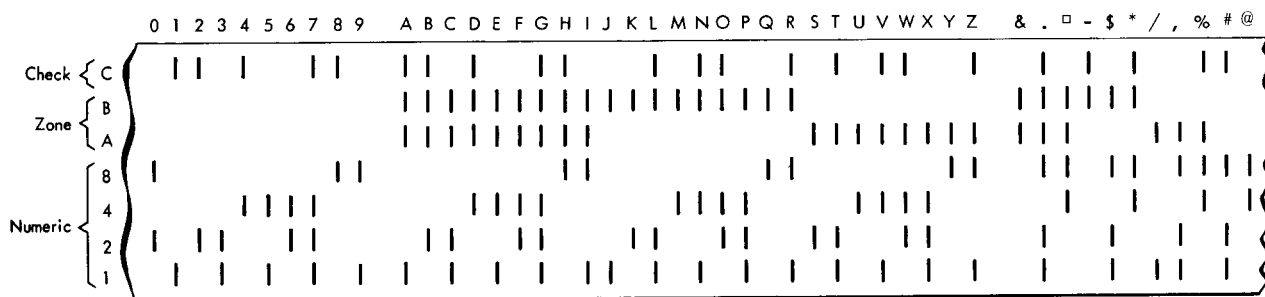


Figure 22. Seven-Bit Alphameric Code on Magnetic Tape

Report Writing Graphics	Programming Languages Graphics	H Code	9 Code	5 Code	14 Code	Report Writing Graphics	Programming Languages Graphics	H Code	9 Code	5 Code	14 Code
Ø (zero)	Ø	0	00	12	12	-	-	11	40	40	40
1	1	1	01	01	01	J	J	11-1	41	41	41
2	2	2	02	02	02	K	K	11-2	42	42	42
3	3	3	03	03	03	L	L	11-3	43	43	43
4	4	4	04	04	04	M	M	11-4	44	44	44
5	5	5	05	05	05	N	N	11-5	45	45	45
6	6	6	06	06	06	O	O	11-6	46	46	46
7	7	7	07	07	07	P	P	11-7	47	47	47
8	8	8	10	10	10	Q	Q	11-8	50	50	50
9	9	9	11	11	11	R	R	11-9	51	51	51
b	⌘	8-2	12	Note	20	!	!	11-0	52	52	52
#	=	8-3	13	13	13	\$	\$	11-8-3	53	53	53
@	'	8-4	14	14	14	*	*	11-8-4	54	54	54
!	:	8-5	15	15	15	]	]	11-8-5	55	55	55
>	>	8-6	16	16	16	;	;	11-8-6	56	56	56
√ (TM)	√	8-7	17	17	17	Δ	Δ	11-8-7	57	57	57
&	+	12	20	60	60	blank	blank	No Punch	60	20	00
A	A	12-1	21	61	61	/	/	0-1	61	21	21
B	B	12-2	22	62	62	S	S	0-2	62	22	22
C	C	12-3	23	63	63	T	T	0-3	63	23	23
D	D	12-4	24	64	64	U	U	0-4	64	24	24
E	E	12-5	25	65	65	V	V	0-5	65	25	25
F	F	12-6	26	66	66	W	W	0-6	66	26	26
G	G	12-7	27	67	67	X	X	0-7	67	27	27
H	H	12-8	30	70	70	Y	Y	0-8	70	30	30
I	I	12-9	31	71	71	Z	Z	0-9	71	31	31
?	?	12-0	32	72	72	‡ (RM)	‡	0-8-2	72	32	32
.	.	12-8-3	33	73	73	,	,	0-8-3	73	33	33
□	)	12-8-4	34	74	74	%	(	0-8-4	74	34	34
[	[	12-8-5	35	75	75	∩	∩	0-8-5	75	35	35
<	<	12-8-6	36	76	76	\	\	0-8-6	76	36	36
‡ (GM)	‡	12-8-7	37	77	77	+++	+++	0-8-7	77	37	37

Note: The octal combination 00 cannot exist in 5 code because it must be written on BCD tape and would be indistinguishable from blank tape. This means there are only 63 possible combinations in 5 code and that 5 code cannot be used directly to represent essentially binary information, such as programs, arithmetic quantities, and so on from the 7040/7044 system.

#### Code Translations

Provision is made in the 7040/7044 system for automatic translation from one code to another, as required, when data are transmitted to or from input/output devices. In some cases, it may be necessary to perform programmed translations (either in the 7040/7044 or in an off-line 1401) to achieve a desired result. Programmed translation is required to maintain compatible card formats when binary information is recorded in H code on cards and it is desired to read or punch the cards both on-line on a card reader and off-line via a card-to-tape or tape-to-card operation. Programmed translation can be avoided if the octal group 12 in 9 code can be omitted, since the information can use BCD tape (rather than binary tape) for off-line operations.

Figure 23. 7040 and 7044 Code Combinations



storage of a 1401 or 1410 and as they exist in the input and output buffers of a 1414 used on the 7040 or 7044 system.

The octal code groups should be interpreted as representations of a six-bit pattern in the order of: (BA8) (421). For example, 101010 equals 52 octal. The entire figure (Figure 23) is in the order of 9 code. This order is the same as the collating sequence on the 7040, 7044, 7090, and 7094 systems. (The collating sequence determines the rank of the characters in compare operations; 00 is low, 77 is high.)

Figure 24 shows BCD characters, both in core storage and as they appear on magnetic tape:

Character	(9 Code)		Character	(5 Code)	
	In Storage	On Tape		In Storage	On Tape
0	00 0000	00 1010	-	10 0000	10 0000
1	00 0001	00 0001	J	10 0001	10 0001
2	00 0010	00 0010	K	10 0010	10 0010
3	00 0011	00 0011	L	10 0011	10 0011
4	00 0100	00 0100	M	10 0100	10 0100
5	00 0101	00 0101	N	10 0101	10 0101
6	00 0110	00 0110	O	10 0110	10 0110
7	00 0111	00 0111	P	10 0111	10 0111
8	00 1000	00 1000	Q	10 1000	10 1000
9	00 1001	00 1001	R	10 1001	10 1001
#	00 1011	00 1011	ō	10 1010	10 1010
@	00 1100	00 1100	\$	10 1011	10 1011
&	01 0000	11 0000	*	10 1100	10 1100
A	01 0001	11 0001	Blank	11 0000	01 0000
B	01 0010	11 0010	/	11 0001	01 0001
C	01 0011	11 0011	S	11 0010	01 0010
D	01 0100	11 0100	T	11 0011	01 0011
E	01 0101	11 0101	U	11 0100	01 0100
F	01 0110	11 0110	V	11 0101	01 0101
G	01 0111	11 0111	W	11 0110	01 0110
H	01 1000	11 1000	X	11 0111	01 0111
I	01 1001	11 1001	Y	11 1000	01 1000
ō	01 1010	11 1010	Z	11 1001	01 1001
.	01 1011	11 1011	±	11 1010	01 1010
□	01 1100	11 1100	,	11 1011	01 1011
			%	11 1100	01 1100

Figure 24. BCD Characters in Storage and on Tape

### Processing Unit Operations

The processing unit controls and supervises the entire computer system and performs the actual arithmetic and logic operations on data. From a functional viewpoint, the processing unit consists of two sections: control and arithmetic-logic.

The control section directs and coordinates all operations called for by instructions. This involves control of input/output devices, entry or removal of information from storage, and routing of data between storage and the arithmetic-logic section. Through the action of the control section, automatic integrated operation of the entire computer system is achieved.

In many ways, the control section can be compared to a telephone exchange. Data transfer paths exist, just

as there are connecting lines between all telephones serviced by a central exchange. The telephone exchange has a means to control the movement of sound pulses from one phone to another, to ring bells, to connect and disconnect circuits, and so on. The path of conversation between one telephone and another is set up by controls in the exchange itself.

In the computer, execution of an instruction involves the opening and closing of many paths or gates for a given operation. The control section can start or stop an input/output device, turn a signal indicator on or off, rewind a tape reel, or direct some process of calculation.

The arithmetic-logic section contains the circuitry to perform arithmetic and logic operations. The arithmetic portion calculates, shifts numbers, sets the algebraic sign of results, compares, and so on. The logic portion carries out the decision-making operations to change the sequence of instruction execution.

### Instructions and Data

The only distinction between instructions and data in core storage is the time when they are brought into the processing unit. Information brought into the processing unit during an instruction cycle is interpreted as an instruction. Information brought into the processing unit during any other type of computer cycle is considered to be data. Consequently, the computer can readily operate on its own instructions if those instructions are supplied as data (that is, if an instruction is brought into the processing unit during any cycle other than an instruction cycle). Also, the computer can be instructed to alter its own instructions according to conditions encountered during the handling of a procedure.

It is this ability to process instructions that provides the almost unlimited flexibility and the so-called logical ability of the stored program system.

In the 7040 and 7044 systems, information (both data and instructions) is handled in fixed groups of 36 positions (bits) each. Each group is called a word. Each position within a word is named with the S (sign) position followed by positions 1 through 35. Computer instructions with an address in the operand part indicate the core storage location to be subjected to some arithmetic or logic operation. This address part, or field, always occupies bit positions 21 through 35 of the word (Figure 25).



Figure 25. Word Address Field

Capacity of the largest core storage available on the 7040 and 7044 systems is 32,768 words of 37 positions each; 36 positions are for data and the 37th is a check bit for the word. The 15-position address field (positions 21-35) is just large enough to hold or indicate the largest core storage address. This address, expressed in the computer's language (code), is simply 15 consecutive 1's (Figure 26).



Figure 26. Word Address with Largest Core Storage Address

Other core storage capacities available with the 7040 and 7044 systems are: 16,384; 8,192; and 4,096 words. In a system with a 16,384-word capacity, the largest address is contained in 14 positions of the address field. The left-most position (position 21) is ignored. Similarly, a 8,192-word system uses 13 positions (23-35), and a 4,096-word system uses 12 positions (24-35) as its address field.

The operation part of most instructions is contained in word positions S, 1-11. Positions 21-35 of the same word would then contain the address of the operand to be used with the instruction. For example, assume that two factors, A and B, are to be added. In the 7040 and 7044 systems, one of these factors is always taken from core storage by the add instruction; the other factor is already in a processing unit register called the accumulator. The accumulator factor must have been placed there by a previous instruction. Figure 27 shows the format of an add instruction when A is the contents of core storage location 00001 and factor B is the contents of the accumulator.



Figure 27. Instruction Format for an Add Instruction

When this instruction is executed, factor A is added to factor B and the resulting sum is returned to the accumulator. Actual computer coding is used (binary code), and the 36 positions are shown in groups of three for easier reading and conversion to the octal number system.

As an example of computer operation, assume that the accumulator contains the number +1. If the number in location 00001 is +2, the result of executing the add instruction is +3. This is shown in Figure 28 (a 0 in the S position indicates a plus number; a 1 indicates a minus number).

### Register

A register is a device capable of receiving information, holding it, and transferring it as directed by control circuits. The electronic components used may be magnetic cores, transistors, or similar components.

Registers are named according to function: an accumulator register accumulates results, a multiplier-quotient register holds either multiplier or quotient, a storage register contains information taken from core storage or sent to core storage, an address register holds the address of a storage location or device, and an instruction register contains the instruction code (operation part) of the instruction being executed (Figure 29).

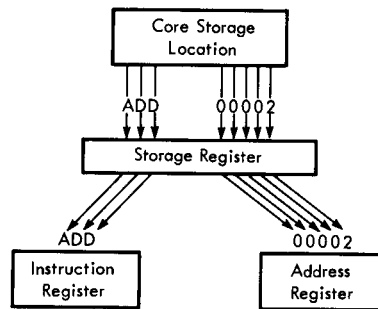


Figure 29. Register Nomenclature and Function

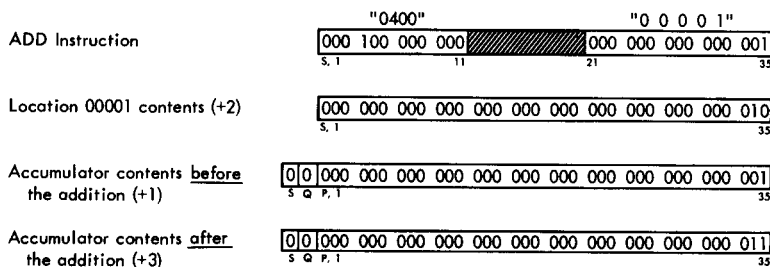


Figure 28. Execution of an Add Instruction

Registers differ in size and use. In some cases, extra register positions are used to detect overflow conditions during an arithmetic operation. The accumulator register is made up of 38 positions; 36 are used for data and two positions (P and Q) are used to remember overflow conditions. If, for example, two 36-bit binary numbers are added, it is possible that the result is a 37-bit answer.

In Figure 30 the accumulator register holds one factor and the other factor, from storage, is in the

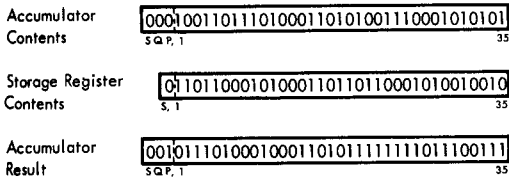


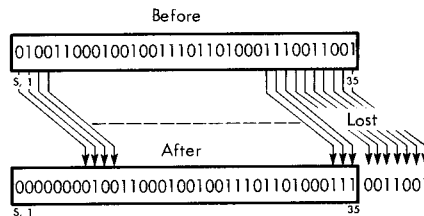
Figure 30. Overflow Condition Resulting from Addition

storage register. The two factors are added and the result is placed back into the accumulator register, where the overflow is indicated by the presence of a 1 bit in the first (P) overflow position. The accumulator might then be shifted right one place and a record kept of the lost low-order bit.

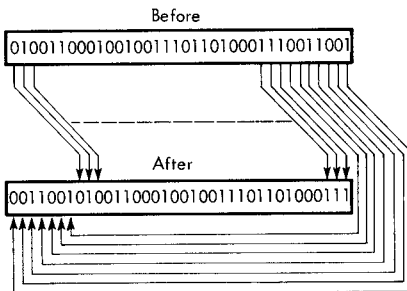
The contents of other registers can be shifted right or left within the register and, in some cases, even between registers. The effect, when shifting from one register to another, is the same as if the two registers were one large register. Figure 31 shows three types of shifting. With shifting within a register, data shifted out of the register may, or may not be lost, depending on the instruction used. With double register shifting, data shifted out of the registers are lost. In the types of shift operations where data loss is possible, vacated positions of the registers are filled with 0's.

In other uses, a register holds data while associated circuits analyze the data. For example, an instruction can be placed in a register, and circuits can determine

Single Register Shifting:  
(Shift right seven places)  
Note: Left-hand positions are filled with zeros; data shifted out of position 35 are lost.



Single Register Shifting:  
(Shift right seven places)  
Note: Data are not lost when shifted out of position 35; the data are re-entered in position 5.



Double Register Shifting:  
(Shift right seven places)  
Note: Data are shifted from position 35 of the first register into position 5 of the second register. Data shifted out of position 35 of the second register are lost. Vacated positions are filled with zeros.

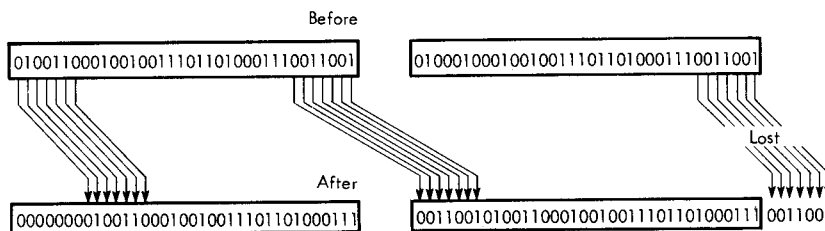


Figure 31. Types of Register Shifting

the operation to be performed and locate the data to be used. Data within specific registers may also be checked for validity.

The main registers of a system, particularly those involved in normal data flow and core storage addressing, have small lights associated with them. These lights are located on the operator's console for visual indication of register contents and various program conditions. If a light is on, a 1 bit is indicated for that position. If the light is off, a 0 bit is indicated.

### Counter

The counter is closely related to a register and usually performs the same functions. In addition, its contents can be increased or decreased by some amount. The action of a counter is related to its design and use within the computer system. Like a register, it may also have visual indicators on the operator's console.

### Adder

The adder receives data from two or more sources, performs addition, and sends the result to a receiving register. Figure 32 shows two positions of an adder circuit with inputs from registers like the accumulator and storage register.

The sum is developed in the adder. A carry from any position is sent to the next higher-order position. The final sum goes to the corresponding position of the receiving register.

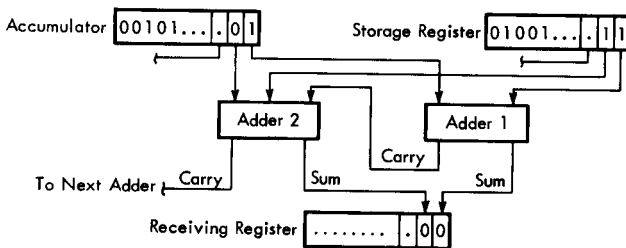


Figure 32. Adders in a Computer System

### Machine Cycles

To receive, interpret, and execute instructions, the central processing unit must operate in a prescribed sequence. The sequence is determined by the specific instruction and is carried out during a fixed interval of timed pulses. These intervals are measured by regular pulses emitted from an electronic clock at frequencies as high as a million or more per second. A fixed number of pulses determines the time of each basic machine cycle.

Within a machine cycle, the computer can perform a specific machine operation. The number of operations required to execute a single instruction depends on the instruction.

All instructions have one instruction (I) cycle and some instructions require only an I cycle for complete execution. Other instructions require both an I and an execute (E) cycle. Various machine operations are thus combined to execute each instruction.

### INSTRUCTION CYCLE

The first cycle required to execute an instruction is called an instruction (I) cycle. The time for this cycle is instruction or I time. During I time:

1. The instruction is taken from a main storage location and brought to the processing unit.
2. The operation part is decoded in an instruction register; this tells the machine what is to be done.
3. The operand is placed in an address register; this tells the machine what it is to work with.
4. The location of the next instruction to be executed is determined.

At the beginning of a program, the instruction counter is set to the address of the first program instruction. This instruction is brought from storage and, while it is being executed, the instruction counter automatically advances (steps) to the location corresponding to the space occupied by the next stored instruction. By the time one instruction is executed, the counter has located the next instruction in the program sequence. The stepping action of the counter is automatic; in other words, when the computer is directed to a series of instructions, it will execute these instructions one after another until instructed to do otherwise.

Assume that an instruction is given to add the contents of storage location 00002 to the contents of the accumulator register. Figure 33 shows the main registers involved and the information flow lines.

I time begins when the instruction counter transfers the location of the instruction to the address register. This instruction is selected from storage and placed in a storage register. From the storage register, the operation part is routed to the instruction register and the

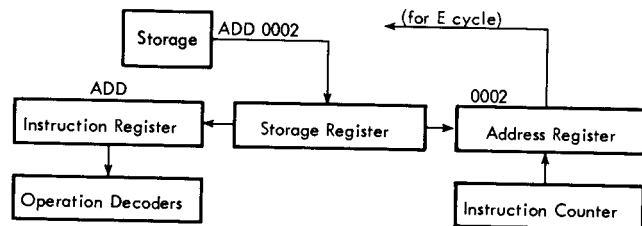


Figure 33. Computer I Cycle Flow Lines

operand to the address register. Operation decoders then condition circuit paths to perform the instruction while the address register locates the operand.

Execution of instructions does not have to proceed sequentially. Certain instructions alter the process of sequential execution unconditionally. In this case, an instruction brought from storage indicates that the next sequential instruction is not to be executed but that one located in another position is next; the normal stepping of the instruction counter is altered accordingly. For instance, the instruction counter can be reset back to the beginning of the program so that the entire program can be repeated for another incoming group of data.

This transfer (branch) to alternative instructions may also be conditional. The computer can be directed to examine some indicating device and then transfer if the indicator is on or off. Such an instruction can say: "Look at the sign of the quantity in the accumulator; if this sign is minus, take the next instruction from location 5000; if the sign is plus, proceed to the next instruction in sequence." The instruction counter is set according to one of the two possible storage locations (5000, or the location of the next instruction in sequence). The logic path followed by the computer (that is, the precise sequence of instructions executed) may be controlled either by unconditional transfers or by a series of conditional tests applied at various points in the program. The arrangement of instructions in storage, however, is not normally altered.

#### EXECUTE CYCLE

I time is usually followed by one or more computer cycles that complete the operation being done by the computer. Execution of an E cycle results in bringing a word into the processing unit from core storage or in taking a word from the processing unit and placing it in core storage. Any word brought into the processing unit during an E cycle is treated as data for the operation decoded by the previous I cycle. Figure 34 shows the data flow following the I time illustrated by Figure 33.

The E-cycle starts by removing from storage the information located at the address (00002) indicated by the address register. This information is placed in the storage register. In this case, the core storage factor is then placed in the adders, together with the number from the accumulator. The contents of the storage register and accumulator are combined in the adders, and the sum is returned to the accumulator.

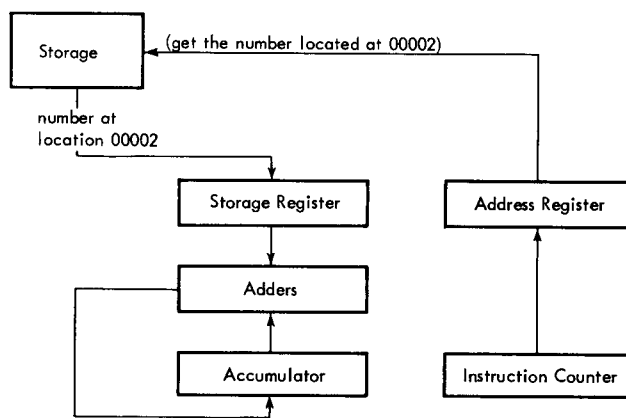


Figure 34. Computer E Cycle Flow Lines

The address register may contain information other than the storage location of data. It can indicate the address of an input/output device or a control function to be performed. The operation part of the instruction tells the computer how to interpret this information.

#### Processing Unit Data Flow

Instruction flow charts are included with many of the instruction descriptions to assist in understanding data and instruction flow through the processing unit. Figure 35 shows a simplified processing unit data flow. In this figure, the positions of the word that are placed in a register or counter are shown below the component.

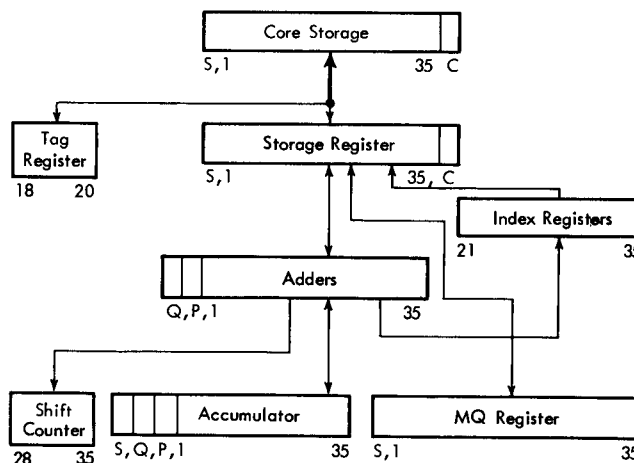


Figure 35. Simplified Processing Unit Data Flow

## Introduction To Programming Systems

A programming system enables the programmer to communicate with the computer in a language closely related to his own language. Thus, the business man might communicate with a business-oriented language and the mathematician might use a language based on mathematical formulas.

The aim of the programming system is to get computer systems into productive operation sooner by freeing the programmer from the intricacies of working in machine language.

Since the IBM 7040/7044 Data Processing Systems are designed as fixed length binary machines, instructions to the computer must be given in binary notation. However, there is an obvious advantage to the programmer in being able to write:

ADD DIVIDENDS TO INCOME

as opposed to writing in machine language:

```
00010100000000000000000000000110000000
0001000000000000000000000000010111000
```

The English language code is easier to learn and use, invites fewer clerical errors, and makes more sense to anyone reading it. These advantages all add up to one thing: high-level languages (related to the programmer's own) allow the programmer to program the problem instead of the machine.

Experience has shown that a computer can be programmed to recognize instructions expressed or written in other languages and to translate those expressions into its own language. This has led to the development of a number of programming languages which are easier to use and understand than the language of the machine.

The first such languages permitted the programmer to write convenient equivalents of machine instructions using symbols (called mnemonics) to represent them. Symbolic instruction representations include: ADD for add, SUB for subtract, DIV for divide, TRA for transfer control, RDS for read, and so on. The computer, acting under control of previously written machine language programs, would then translate these symbolic instructions into equivalent machine instructions, which could then be used in solving the actual problem.

These first languages resulted in a one-for-one translation. That is, each instruction written in the programming language was translated into one machine language instruction. For example, the instruction:

ADD    184

would produce the machine language instruction:

```
0001000000000000000000000000010111000
```

Later, "macro-instructions" were developed. That is, single programmer language instructions could be used to produce a whole series of machine instructions. This development greatly increased the power of programming languages. The art of programming has progressed to a point at which it is possible to give directions to a computer by writing statements and sentences in a language which is based on, and which can be read in the same way as, English itself or even mathematical formulas.

The translation feature of the machine language program is perhaps the most important feature, but not the only one. The computer instructions needed to produce a given result must be executed in a given sequence. If an addition is to be performed, one of the values involved must be in the computer before the add instruction itself is executed. This is normally accomplished with an operation called CLA, "Clear and Add." After this operation is executed, the add operation may then be executed. The two-instruction sequence is shown in both machine and symbolic language as:

Machine	Symbolic
00010100000000000000000000000101111111	CLA    383
00010000000000000000000000000110000000	ADD    384

Each final machine language instruction must be assigned a particular location in core storage. If the CLA instruction is to be assigned a location of 100 (its precise slot in core storage), and the ADD instruction is to immediately follow it, the location of the ADD instruction must be 101. Therefore, the location of each instruction must be known precisely. It is, in effect, the "name" of the instruction. If an additional instruction is to be inserted in a program of many instructions, every instruction from the point of insertion must have its previously assigned location changed. Since most programs undergo changing or up-dating, instruction location assignment becomes a tedious but necessary part of programming. The solution, of course, is to have the translating program do the actual assignment of instruction locations in addition to its translating function. The programmer need simply tell the translating program the desired location of the first instruction and succeeding instructions are assigned sequentially ascending locations.

The advantage of expressing a problem in symbolic language over machine language should now be evi-

dent. This symbolism may be carried one step further by using symbolic data addresses as well as symbolic operation codes. The translating program can then be designed to translate and assign these symbols to actual core storage locations. Using the same instructions as before, assume that the two values to be added are expressed as values "A" and "B". Of course, in both methods the values must have been previously placed in core storage, but the problem can now be stated as in Figure 36.

Instruction Location	Instruction	
	Operation Part	Address Part
	CLA	A
	ADD	B

Figure 36. Symbolic Operations and Addresses

If we now tell the translating-assigning program we want the first instruction placed at core storage location 100, the program shown in Figure 37 would result (the program is expressed with symbolic operation codes and decimal addresses and locations, instead of machine language, for better understanding).

Instruction Location	Instruction	
	Operation Part	Address Part
100	CLA	102
101	ADD	103
102	Value A	
103	Value B	

Figure 37. Assigned Addresses and Locations

In a basic sense, the translating-assigning program is called a "processor program," or, more simply, the processor. In normal operation, the processor is entered into the computer system and placed in some type of storage. Next, the instructions, prepared by the programmer to accomplish a particular job as coded in his language, are fed into the computer. The computer then, in translating the programmer's instructions into machine language instructions, writes its own program. The translated machine instructions are placed in core storage and form the actual job program.

### Operation

A processor is made up of at least two parts: a language, with associated rules of grammar; and a machine language program, whose main function is to translate the language of the programmer into machine language.

The input to a processor is called the source program. This is written by the programmer in the language of the programming system (processor language) and

states the requirements of the problem and the method of solution. Before the programmer writes his source program, he must have completely analyzed and defined the problem.

The output from the processor is the object program, the translation of the source program from the programmer's language to the language of the computer system on which the program will be used. Figure 38 shows the basic procedure for producing an object program; this is called the assembly or compiling run. In this case, the assembly (processor) program has been previously recorded on a reel of magnetic tape and is available to the computer for the assembly process.

The source card deck may be fed directly to the computer or recorded on another magnetic tape reel in an off-line card-to-tape operation. Off-line means a separate operation not controlled by the computer and may be considered as a preparatory or set-up operation. The information from each source card is translated by the assembly program instructions into object (machine) instructions. Each object instruction is then placed in storage. When all source information has been translated and assembled, the resulting object program is sent from the computer signalling the end of the assembly run.

The object program is printed (program listing) and an object program card deck is produced. This operation, as with source program input, may be produced directly from the computer or by off-line tape-to-card and tape-to-printer operations. Note that both the program listing and object card deck are referred back to the programmer for modification or correction if necessary.

Once the object program is satisfactory, the execution run may be started (Figure 39). Fed into the computer with the object program are the data required to solve this particular problem. As with Figure 38, the information is fed directly to the computer from card decks or from magnetic tape reels prepared off-line. Subroutines (standard programs used with many problems) together with rate tables and other constant factors may also be available to the computer to assist in the execution of this problem. Figure 39 shows the use of a subroutine tape for this purpose.

After the problem is executed (solved) by the computer, the result (output) may be recorded on magnetic tape for an off-line printing operation or printed results may be received directly from the computer.

A proven object program may be used time after time, with varying problem data, to produce periodic results — such as production type programs of payroll or inventory; or to produce different results to assist the designer seeking an optimum design — such as the best

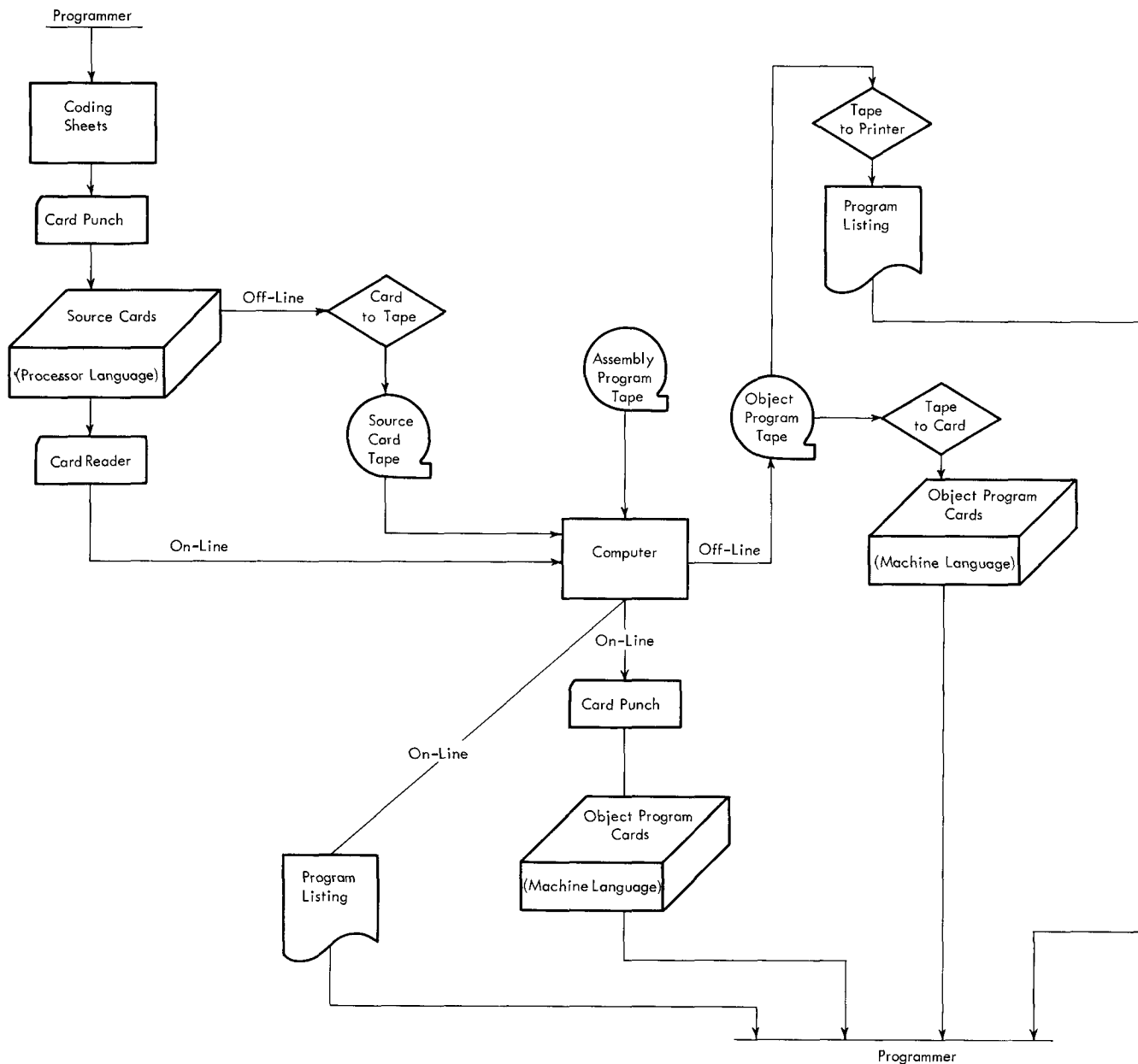


Figure 38. Assembly Process to Object Program

wing air foil, or the most efficient placement of steam pipes within a boiler, considering all variables for each application.

### COBOL System

The COBOL (Common Business Oriented Language) system, unlike the first programming languages, is "problem oriented." That is, the language itself, and the techniques for using it, are conceived in terms of the problems to be solved and the results to be obtained; not, for the most part, in terms of the technical features of the computer. Of course, each problem

must still be solved by technical means; it is still necessary to produce a machine-language program before a problem can be solved. However, the language written by a COBOL programmer bears little resemblance to machine language, and the programmer has little direct concern with the method by which the COBOL language program is translated into machine language.

A simple example will best illustrate the basic principles of the problem-oriented type of programming system. Assume we wish to increase the value of an item called INCOME by the value of an item called DIVIDENDS. The COBOL language allows us to specify this addition by writing the following sentence:

ADD DIVIDENDS TO INCOME.



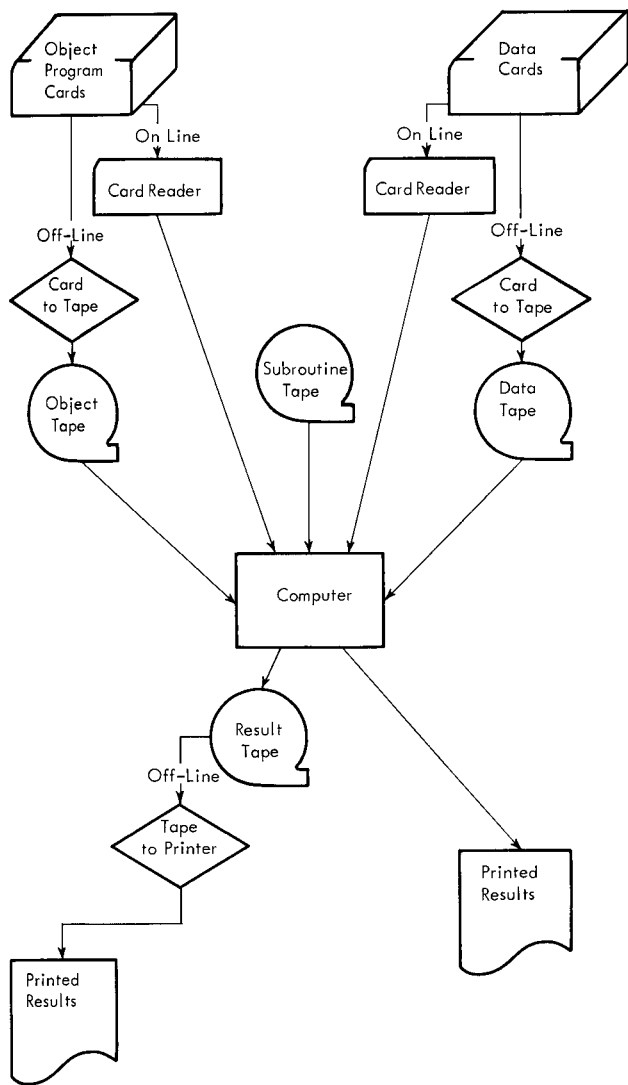


Figure 39. Execution Process Using Object Program

Before the processor can interpret this sentence, however, it must be given certain information. For example, the programmer will have to write the names *DIVIDENDS* and *INCOME* in a special part of the program called the "data division." Here, facts about the data represented by those names, such as maximum size, how the data is expressed, and so on, are stated.

When the processor encounters the sentence, it has access to certain information that will aid it in translating the sentence. In addition, it will be able to obtain certain information "built into" the processor itself. (The reader should note, however, that the exact procedure will vary from machine to machine and that, in any case, the programmer is not directly concerned with the details.)

First, the processor examines the word *ADD*. It consults a special list of words that have clearly defined meanings in the COBOL language. This list is a part of

the processor. If *ADD* is one of these words, the processor interprets it to mean that it must insert into the object program the machine instruction (or instructions) necessary to perform an addition.

The processor then examines the word *DIVIDENDS*. Since it can obtain certain information about *DIVIDENDS*, it will know where and how this information is to be stored in the computer, and it will insert into the object program the instructions needed to locate and obtain the data.

When the processor encounters the word *TO*, it again consults the special word list. In this case, it finds that *TO* directs it to the value of *INCOME* which is to be increased as a result of the addition.

The processor must now examine the word *INCOME*. Again it has access to certain information about this word, and, as a result, it is able to place in the object program the instructions necessary in locating and using *INCOME* data.

We have indicated that the programmer placed a period (.) after the word *INCOME*, just as he would in terminating an English-language sentence. The effect of the period on the COBOL processor is quite similar. It tells the processor that it has reached the last word to which the verb *ADD* applies.

The previously described steps are performed by the processor in creating the object program. They might not be performed in exactly this way or in the same sequence, because machines vary and because each processor is adapted to a particular machine. However, regardless of the machine, the same COBOL-language sentence produces machine instructions that will cause the object program to add together the values *DIVIDENDS* and *INCOME*.

## FORTRAN System

The FORTRAN (Formula Translation) system is very similar in concept to the COBOL system. One of the main differences is in the language the programmer uses to express his source program. Where business English is used by COBOL, mathematical language is used with FORTRAN. The effect of the COBOL sentence:

*ADD DIVIDENDS TO INCOME*

could be achieved by the FORTRAN statement:

*INCOME = DIVIDENDS + INCOME*

However, FORTRAN processors for some machines might insist that the words be abbreviated to something like:

*INCO = DIV + INCO*

This would depend on the individual machine FORTRAN processor. The statement, in effect, tells the processor to insert the necessary instructions into the object program to make the *INCOME* data location equal to the

DIVIDEND data added to the present INCOME data. Note that the computer is not merely instructed to find the value of INCOME, but is also told where to put the result of the addition after it is performed. If the original INCOME field (in core storage) contained 10000, and the DIVIDEND field contained 15, the original INCOME field would be replaced by 10015 after the operation has been executed.

If this result is not desired, the programmer could change the statement to:

INCOME 1 = DIVIDENDS + INCOME

With this change, a new INCOME 1 data field would be generated in core storage, the result of the addition would be placed there, and the original INCOME field would remain unchanged.

Since only a few of the many features of the COBOL and FORTRAN systems have been discussed, the reader is referred to the *COBOL General Information Manual*, Form F28-8053, or the *FORTRAN General Information Manual*, Form F28-8074, for additional information.

### Programming System Segment Relationship

Thus far, the terms Symbolic Language, Programming System, Source and Object Programs, and Processor have been used. In actual operation these terms are expanded to:

1. *Programming System*: Any method of programming problems, other than machine language, that consists of a language and its associated processor(s).

2. *Symbolic Language*: Any collection of symbols used in programming to represent operation codes, functions, addresses, with rules of usage.

3. *Processor*: A machine language program that performs the functions necessary to convert a source program into the desired object program.

4. *Source Program*: A program coded in other than machine language that must be translated into machine language before being used.

5. *Object Program*: A program coded in machine language for use by the computer.

6. *Compiler*: A translation program that translates macro-instructions of a symbolic program into one-for-one symbolic instructions, and then passes the entire set of instructions to an assembly program for final translation.

7. *Generator*: A machine language program used during compiling to produce symbolically coded (one-for-one) instructions that will perform the operation called for by the symbolic coding of the source program.

8. *Assembly Program*: A translation program that substitutes binary coding for symbolic instructions, may assign storage locations, and performs other activity

necessary to produce an object program directly loadable into the computer. This object program may be self-loading or, in some systems, a load program is needed.

Figure 40 shows the segment relationship in programming systems.

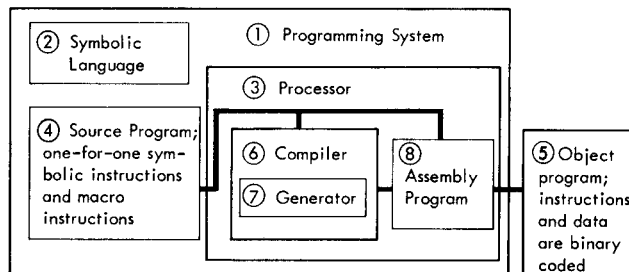


Figure 40. Programming System Segment Relationship

### Program Checkout

After successful translation and assembly of a source program, execution of the resultant machine language object program with test data occurs. This is done to assure that the program does not have logical errors and that it is capable of producing a right answer when using test data. Two results are possible. The first and hopefully only result is that the problem (for which the program was written) can now be attempted with real data. The second result — the test run does not function properly — may occur because of many things. The most frequent cause is that the source program has been improperly or incompletely stated.

Mistakes by the programmer are more difficult to avoid than might be expected. It is, in fact, a rare program that works correctly the first time it is tried with test data. In most cases, several test runs must be made before all mistakes are found and corrected. The programming system itself finds most of the obvious mistakes during the translation and assembly run. Such things as calling for a storage location by a name when that name has not been defined, attempting to perform integer arithmetic on floating point data (or the reverse), lack of defined alternative paths on testing operations, and keypunching errors of all kinds are detected and noted during the assembly and translation run of the program.

Computer mistakes are usually obvious. Built-in detection circuits will normally reflect the kind of mistake the computer has made by turning on an indicator and stopping the computer. Detection and classification of the mistakes a programmer can make are, however, many times more complex.

## Testing Techniques

As previously stated, a computer program may be expressed in machine, symbolic, or one of the problem-oriented languages such as FORTRAN or COBOL. Source program coding is harder and more error prone when machine language is used, but becomes progressively easier with symbolic and higher level languages. These circumstances are reversed when source program debugging is required. That is, it is easier to debug a symbolic language program than a FORTRAN or COBOL program. The main reason for this is that the symbolic program results in a one-for-one translation (one machine language instruction for each symbolic language instruction), whereas high-level languages usually result in a many-for-one translation.

Many techniques exist to assist the programmer during the check out phase of his work. Each has its own advantages and disadvantages. The one to be used for a particular problem will depend upon the programmer's thoughts as to what area of his program is in trouble and how extensive the trouble is. Techniques that involve extensive use of switches on the operator's console are very wasteful of computer time and are not recommended.

## Storage Printout

This type of utility program (routine) is most efficient from a machine standpoint because practically the entire contents of storage, plus the contents of working computer registers and the condition of indicators and switches, may be presented in printed form. Normally, the register contents and condition of indicators and switches are printed first. The contents of storage are then printed. Each line of printing representing storage begins with the starting location of that line expressed in octal format. Seven complete word locations are printed on each line. The print (dump) routine sometimes has provisions for dumping one or more selected blocks of storage instead of all of it. It may also have the ability to restore the dumped blocks of storage back into their original locations.

## Tracing

If visually checking a storage print-out fails to reveal the program difficulty, a technique called "tracing" may be used. The trace technique usually involves an interpretive routine and, therefore, executes a number of instructions for each program instruction being traced. The print-out received while tracing normally includes: the location of the instruction being executed, the instruction being executed, and the contents of the working registers after the instruction has been executed. The printing of each instruction execution in a program would result in excessive machine time

and should be used only when all other methods fail to reveal the program trouble.

The basic tracing technique may be revised, however, and only selected storage locations can be printed when program execution reaches a specified point in the program. With this variation, a "snapshot" can be obtained of a particular part of the program under particular conditions. For example, the trace and resultant print-out can be specified to occur only when the program executes a transfer instruction. A whole series of "snapshots" will then be obtained showing the execution path through the program. Likewise, only those instructions which altered the normal execution path can be "snapshot," to show the exception paths the program has executed.

## Summary

Successful program check out depends on many things. The time consumed by this necessary but frustrating phase of programming may be lessened if certain basic rules are followed:

1. Document your program wherever possible so that you (and anyone else) will know what you intended to accomplish by a given program step.
2. Check the source program cards against the documentation before an assembly and test run is attempted. This point cannot be overemphasized.
3. Leave space in your program for insertion of testing or printing routines that may be used in the test run of the program. Program space is useful also if changes have to be made.
4. Be aware of the debugging techniques available, and know how best to use them; avoid becoming a slave to one technique, excluding all others.
5. Be absolutely sure that the program does what it is supposed to do and nothing more.
6. A successful test run does not insure that the program will run to completion with actual data. Actual data may be too large for the storage area assigned to it, too slow to be properly processed by the program, or may not be in the planned data format.

## Input/Output Control Systems

While macro instructions save much labor, the problem of organizing input/output operations in a complex application could still involve considerable work on the part of even the most expert programmer. From a standpoint of simplicity, it is far easier to work with one record at a time; that is, read a record, process the record, and then write the resulting record. However, efficient utilization of tape or disk systems requires that records be grouped both on input and output and that the processing of records be scheduled to best use the available computing time.

To solve this and other problems, the concept of input/output control systems (IOCS) was developed. Basically, adding IOCS to a programming system makes it possible for the programmer to think of his problem as a simple sequential operation. Given a description of how the input and output files are organized, the processor associated with the IOCS takes care of all the machine language coding necessary to read and write tape, card, or disk storage records.

It is important to recognize that the IOCS statements, which give the programmer the facility of using these techniques for input/output programming, are part of the total language for an individual system. The importance of these subroutines and their relatively recent inclusion in programming systems have led many to discuss them as a separate subject. There can be no argument with this sort of discussion as long as we keep in focus the entire programming problem and the relative place of input/output control systems.

The use of an IOCS, then, enables the programmer to divorce himself almost completely from the physical requirements of the data, the recording media on which the data are written, and the input/output devices on which the media are mounted, and permits him to concentrate most of his efforts upon the processing of the data.

With disk storage attached to data processing systems, additional complexities of input/output programming have been introduced. Because of the random access nature of these devices, proper scheduling becomes even more important and more difficult.

Where a tape IOCS can use the serial nature of tape files to call in the next block from the tape before it is requested by the user's program, this is not always possible when using a disk. Here, the "next" record of the file may be physically located anywhere in the disk storage, and several "logical" records may share the same physical disk record. Many techniques exist for solving this problem. Some are quite simple; others are very complicated. Several of the latter involve segmenting the user's program into two or more subprograms. Each subprogram can process one type of record or locate the new record of a given type. The IOCS, then, can enter these subprograms in a more or less random sequence, depending on which of many records being sought on the file is found first. This is actually a simple form of multiprogramming, where several different logical programs perform their computation in a sequence partially dictated by a master scheduling routine.

Important features found in most of the input/output control systems in use are listed below. No one IOCS contains all features listed, but all of them utilize many of the procedures. Features not in universal usage are included to show the great versatility of these systems.

Input/output control systems have grown past the point of merely handling the normal input and output requests and are becoming an integral part of the entire operating system for a data processing system, in some cases handling the manipulation of data internally as well as to or from the input/output devices.

### **Input/Output Scheduling**

Some computers handle input/output in a serial, synchronous fashion. No computing can be done until an input/output operation is completed and, conversely, no I/O can be done while the central processor is engaged in computation. Other computers, however, achieve simultaneous input/output and computing operation by simply allowing the central processor to continue with its operation while the input/output device locates data or reads it into or out of the main storage of the system. This simultaneous (asynchronous) input/output for all types of input/output equipment helps greatly to prevent unnecessarily delaying the central processor while information is being read into or out of main storage.

The use of an IOCS, then, allows the programmer to easily make use of the complex asynchronous input/output devices that permit a modern data processing system to operate efficiently.

### **Blocking and Deblocking of Records**

High density tape or disk storage units become relatively inefficient when used to record short blocks of information. When recording 80 character blocks, for example, over three-quarters of the file contains no useful information; instead, it is made up of end-of-record gaps. By grouping together or *blocking* a number of such short records, all but one of the useless end-of-record gaps can be eliminated. The result is that a given length of tape contains several times as much information as before. Since the tape passes through the tape unit at a fixed rate, the tape unit now spends more of its time reading useful information and less time spacing over end-of-record gaps. The end result is a higher effective input/output data rate.

Note that this technique fails if the records are not requested often enough to keep the tape unit in continuous operation. In this case, the speed of the central processor becomes the limiting factor and the program is said to be process limited. If the reverse is true, the program is said to be input/output limited, and blocking may be used to decrease the time required to read an average logical record.

Since input/output units usually require that the entire physical block be read or written once transmission is started (there is no way to stop tape motion in the middle of a block), it is desirable to collect

together all records to be written as one block and, conversely, on input, to unpack or deblock such a physical block into its many logical records and release them to the processing program as requested by it.

### **Standard Error Correction and Unusual Condition Routines**

Many conditions met in performing input/output are exceptions to the normal case of simply reading or writing a record. The programmer does not wish to concern himself with all of these eventualities each time he makes an I/O request. For example, he should not have to perform a test each time a file is referenced to determine if the end of the file has been reached. Doing so makes the infrequent end-of-file condition require as much programming, perhaps, as the normal reading of the record. Many unusual or exceptional conditions are of a general nature and, as such, can be handled by common routines within an IOCS.

Listed below are a few of the exceptional conditions detected or handled by input/output control systems:

#### **ERROR CORRECTION PROCEDURES**

If transmission to or from an input/output device is not successful the first time it is attempted, certain techniques can be used to attempt to clear the failure and allow the program to continue uninterrupted. Such standard error correction routines might involve an attempt to erase a record recorded incorrectly on tape and to rewrite the record correctly. Obviously, if the rewrite is successful, the programmer need not be concerned and need not provide additional instructions to handle the result.

If the repeated erase and rewrite are not successful in clearing the failure, only then does the machine operator or the program need to be informed of the uncorrectable error. Thus, most errors can be automatically corrected without any additional programming being required.

#### **END-OF-REEL AND END-OF-FILE PROCEDURES**

When all data records on a single reel of tape are processed, the tape is said to be at end-of-reel. If, in addition, all records of the file, which can consist of more than one reel, are processed, the file is said to be at end-of-file. Obviously, if an end-of-file condition is met, the processing of that file is complete and the user's program must be informed of this fact.

#### **IMPROPER LENGTH RECORD PROCEDURES**

If a record is read that, through malfunction or programming error, is not of correct length, this condition must be detected and corrective action taken.

If the error is such that the system cannot continue processing the current job, automatic transition to the next job can be initiated or the system may be stopped after informing the operator of the nature of the error. In some cases, it is enough to inform the user's program of the condition and allow it to make the decision as to how the condition is to be handled.

### **Tape Labeling**

The maintenance of a large library of tapes containing data costing thousands of dollars to generate imposes a large responsibility of preserving the integrity of the data. A careless operator, who inadvertently mounts a master tape containing valuable data and allows the tape to be written upon, can cause almost complete collapse of the application using this master tape (writing on tape automatically erases previously recorded information).

To insure accurate library maintenance, a technique of tape labeling has been developed. This technique consists of recording, as the first block of information on each reel, a header label containing information that uniquely identifies the reel to the user's program or the IOCS label checking routine. By comparing the desired reel identification against the information recorded on the reel, correct reel mounting can be insured and file integrity preserved.

## IBM 7040/7044 Programming Systems Programs

The IBM 7040/7044 Operating System (IBSYS) is an integrated set of systems, coordinated by the IBSYS Basic Monitor and using the 7040/7044 Input/Output Control System. The basic monitor provides continuous operation during a sequence of jobs, each of which might involve a different system. The systems that work with the Basic Monitor include:

**IBEDT** A systems library editor that maintains the system library.

**IOCS** A control system for efficient scheduling of input and output.

**IBSRT** A generalized sorting program to sort and merge data.

**IBJOB** A processor program containing the following components:

**IBJOB Monitor:** Supervises the execution of the compilers, assembly program, and loader.

**IBLDR Loader:** Processes and combines programs produced by IBMAP to form one binary object program.

**IBLIB Subroutine Library:** Contains routines that will be loaded if required for object program generation.

**IBMAP Macro Assembly Program:** Processes programs written in the MAP language (a machine-oriented language with macro facilities) and the internal language programs produced by the COBOL and FORTRAN compilers. IBMAP produces from each compilation a binary program deck that retains enough symbolic content to enable communication with previously compiled program decks.

**IBFTC FORTRAN Compiler:** Processes programs written in the FORTRAN IV language (a scientifically oriented language) and produces input to IBMAP.

**IBCBC COBOL Compiler:** Processes programs written in the COBOL language (a commercially oriented language) and produces input to IBMAP.

Figure 41 shows the relationship between IBSYS operating system components. The input/output control system (IOCS) and the IBSRT, IBJOB, and IBEDT monitors communicate directly with the basic monitor. After control is taken by IBSRT, IBJOB, or IBEDT, the individual monitor of that component controls the system until a new control card is encountered.

The IBM 7040/7044 Operating System contributes to the flexibility and economy of the computer installation by:

1. Significantly reducing machine time and human handling when processing a stack of jobs. Specifically,

the system reduces the number of tape reel changes, provides a convenient method of updating and modifying all system programs, allows storing all systems programs in one storage device, makes possible continuous machine operation with a minimum of operator intervention, and uses control cards to specify input/output devices available to the basic monitor.

2. Providing for adaptation of IBM programming systems to a wide range of input/output devices. This enables a particular installation to gain immediate benefits by the addition of new input/output devices according to its requirements. An installation need not be card, tape, or disk oriented.

3. Allowing the user to maintain up-to-date source language statements of his jobs by modifying only segments of a program in FORTRAN or COBOL language. A small section of a program may be re-compiled without the necessity of re-compiling the whole program.

Figure 42 shows the operation of the IBJOB Processor on source language programs of different types.

The operation of the IBM Operating System (IBSYS) is automatic; once an input reel is mounted it should not be necessary for the computer to be idle until the output reel is dismounted, provided enough input/output devices are available. It should not be necessary for the operator to take any action other than dismounting unloaded reels and replacing them with reels to be used later in the job or on succeeding jobs.

The operation of each phase of IBSYS is directed by control cards, but the programmer has to use only the

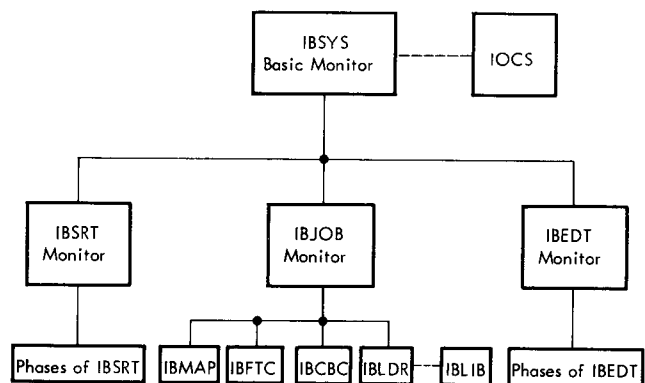


Figure 41. IBSYS Operating System Components

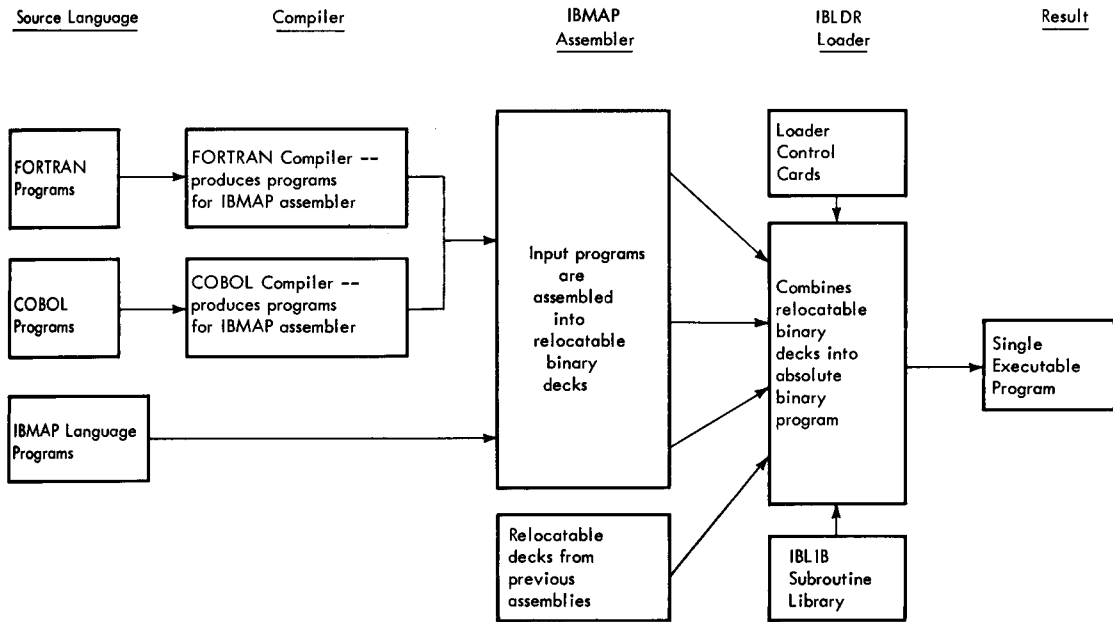


Figure 42. IBLJOB Processor Flow Chart, Source Programs

parts of the operating system that apply to his current job. Basic monitor control cards are ordinarily not used by the programmer; they are the concern of the machine operator. Thus, the programmer concerns himself with a comparatively small number of control cards to run any one job.

### Over-all Operation

The basic monitor (IBSYS) acts as an intersystem monitor that calls the appropriate submonitor according to control card specifications. A portion of IBSYS remains in core storage at all times and permits return of control to the basic monitor, reference to the input/output control system, and loading of the programming systems into core storage. This portion of IBSYS contains a set of common system routines and subroutines and a communications region where common information shared by the programming systems is maintained.

A supervisory portion of the basic monitor is called in when required to transfer control between system monitors, to initialize the first portion of IBSYS, to change the machine environment, and to assign external storage devices to logical input/output functions.

Figure 43 shows a typical input to the IBSYS operating system. The input consists of a data card (handled

by the basic monitor), two jobs to be processed by IBLJOB, followed by the control cards for an IBSRT machine run. The first job to be processed by IBLJOB is a program consisting of two FORTRAN source decks, two MAP source decks, a binary deck (from a previous compilation), and data cards. The second job is unspecified.

When the basic monitor (which is already in core storage) encounters the \$IBJOB control card, it transfers program control to the IBLJOB Monitor. The \$ symbol specifies that this card is a control card. The \$IBJOB card is recognized by the IBLJOB monitor, as well as by the basic monitor, and indicates the beginning of a new job. Thus, the IBLJOB monitor will supervise job-to-job transition until it encounters a \$IBSYS control card, which indicates that the next operation is not within IBLJOB's operating scope. When this occurs, program control is returned to the basic monitor.

When the basic monitor encounters a \$IBSRT control card, it calls in the sort monitor (supervisory portion of the generalized sorting program) and transfers control to it. The sort monitor controls execution of the sort, according to specifications on the control card. Successive sorting and merging jobs can be handled by the sort monitor without relinquishing control to the basic monitor. Control is returned to the IBSYS basic monitor when the next \$IBSYS control card is encountered.

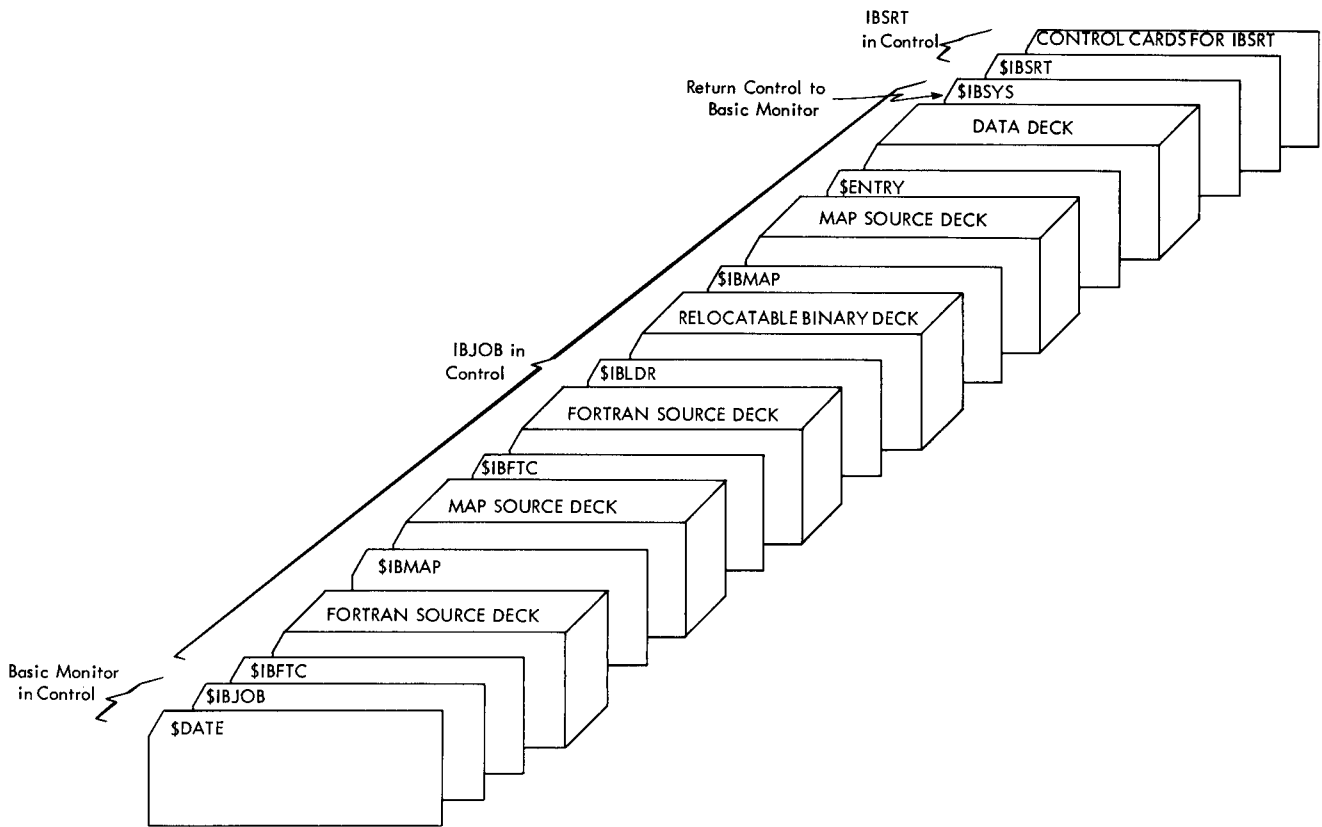


Figure 43. Typical Input for the IBSYS Operating System

### Macro Assembly Program

The IBM 7040/7044 Macro Assembly Program (IBMAP) is a versatile, general purpose assembler for the 7040 and 7044 Data Processing Systems. It accepts source language statements written in the MAP language.

A MAP symbolic instruction consists of four major divisions: location field, operation field, variable field, and comments field. A portion of the coding form used is shown below.

Problem								
Coder						Date		Page
Location	Operation			Address, Tag, Decrement/Count		Comments		
1	2	6	7	8				

The location field normally contains a name that other instructions may refer to when that instruction is to be executed, the operation field contains the machine operation (or pseudo-operation), and the

variable field normally contains the location of the operand. The comments field is for the convenience of the programmer and plays no part in directing the computer.

Symbolic instructions, expressed in the MAP language, are punched one instruction per card. The card format is:

CARD COLUMNS	USE
1 through 6	<i>Location Field:</i> May be blank.
7	Not used, must always be blank.
8 through 14	<i>Operation Field:</i> Starts in column 8 and is three to seven characters long. A blank column must separate the operation and variable fields.
12 through 71	<i>Variable Field:</i> May begin in column 12 (if operation field is three characters), but may not begin after column 16. Each user normally assigns a fixed column as the beginning of the variable field for all instructions. The variable and comment fields must be separated by a blank column.
17 through 71	<i>Comments Field:</i> Follows the blank column after the variable field and may extend up to column 71.
73 through 80	Normally used for identification of the program being assembled.



## Macro Assembly Program Language

In writing symbolic instructions, the programmer is concerned with building expressions to represent the address, tag, and decrement portions of the machine instruction.

The smallest component of an expression is an element, which is either a single symbol or a single integer. When elements are combined with operators (symbols representing machine operations) a term is formed. A term may consist of a single element, two elements separated by an operator such as the \* or / character (\* represents multiply and / is the divide character), three elements separated by two operators, and so on. A term must begin with an element and end with an element. Two operators or two elements in succession are not allowed.

In addition to being an operator, the asterisk is also an element. In this use, the asterisk stands for the location of the instruction in which it appears. Thus, the element \* will have different values in different instructions. There is no ambiguity between this use of the asterisk and its use to denote multiplication, because the position of the asterisk always makes clear what is meant.

An expression is made up of terms separated by the + or - operators, (+ means add and - means subtract). An expression may consist of a single term, two terms separated by + or -, three terms separated by two operators, and so on. The programmer may not write two operators in succession or two terms in succession, but an expression may begin with + or -.

An expression is terminated by a comma symbol or, for the last expression of a statement, by a blank column. A negative expression is represented in 2's complement notation (See "Indexing Concept"). A null expression is an expression that is indicated as being present but has no value. It can occur:

1. When an assembly scan encounters a comma rather than the first element of an expression. The comma shows that a null expression is indicated. Two consecutive commas indicate a null expression, or a comma as the first character of the variable field indicates that the first expression is null.

2. When a scan encounters a blank following a comma. This character combination indicates that the last expression of the statement is a null expression.

### Operation Codes

The IBMAP program recognizes all 7040/7044 machine operation codes. Instructions consist of:

1. A symbol or blanks in the location field.
2. The appropriate operation code in the operation field.

3. Address, tag, and decrement (or count) subfields appearing in the variable field, each of which may be a symbolic expression.

### Literals

Often a programmer wishes to refer to a word containing a constant. For example, if he wishes to add the number 1 to the contents of the accumulator, he must have somewhere in storage a word containing the number 1. Pseudo-operations are provided to allow introduction of data words and constants into the program, but often this introduction is more easily accomplished by the use of a literal.

The appearance of a literal directs the assembler to prepare a constant equivalent in value to the contents of the literal subfield, store this constant in a location at the end of the program, and replace the address field of the instruction containing the literal with the address of the constant thus generated. Three types of literals are permitted: decimal, octal, and alphanumeric.

#### DECIMAL LITERALS

A decimal literal consists of the = symbol followed by a decimal data item. For example, the instruction `MPY = -3` means "multiply the contents of the MQ by the decimal number -3." (That is, multiply the contents of the MQ by the contents of a storage location that contains -3.)

Three types of decimal data items are recognized:

*Decimal Integer.* A decimal integer is composed of one or more digits, and may be preceded by a plus or minus sign. A decimal integer is distinguished from other types of decimal data items by the absence of the letter B, the letter E, and the decimal point.

*Floating-Point Number.* A floating-point number has two components:

1. The *principal part* is a decimal number written with a decimal point. The decimal point may appear at the beginning, at the end, or within the principal part, or it may be omitted if the exponent part is present. If omitted, the decimal point is assumed to be at the right end of the principal part.
2. The *exponent part* consists of the letter E followed by a signed or unsigned decimal integer. The exponent part must follow the principal part; it may be omitted if the principal part contains a decimal point.

A floating-point number is distinguished from a decimal integer by the presence of either a decimal point or the letter E (or both). It is distinguished from a fixed-point number by the absence of the letter B.

*Fixed-point number.* A fixed-point number has three components:

1. The *principal part* is a decimal number written with or without a decimal point. The decimal point may appear at the beginning, at the end, or within the principal part, or it may be omitted. If omitted, the decimal point is assumed to be at the right end of the principal part.
2. The *exponent part* consists of the letter E followed by a signed or unsigned decimal integer. The exponent part may be absent; if present, it must follow the principal part, and may precede or follow the binary-place part.
3. The *binary-place part* consists of the letter B followed by a signed or unsigned decimal integer. The binary-place part must be present in a fixed-point number and must follow the principal part. If the number has an exponent part, the binary-place part may precede or follow the exponent part. A fixed-point number is distinguished from other types of decimal data items by the presence of the letter B.

Literals are considered to be single precision numbers unless two E's (EE) appear in the exponent of a floating-point number. Double precision numbers are stored in consecutive storage locations with the high-order part first. (See "Double Precision Floating Point Instructions.")

#### OCTAL LITERALS

An octal literal consists of the character =, followed by the letter O, followed by a signed or unsigned octal integer. For example, the instruction `ADD =O37` means "add to the contents of the accumulator the contents of a core storage location that has 00000000037<sub>8</sub>."

#### ALPHAMERIC LITERALS

An alphameric literal consists of the character =, followed by the letter H, followed by six BCD characters. For example, the instruction `LDQ =H12AB` would load into the MQ the contents of a core location that contains 010221226060<sub>8</sub>. When fewer than six BCD characters are specified, the unspecified characters are assumed to be BCD blank characters.

### Data-Generating Operations

These pseudo-operations (OCT, DEC, and BCI) may be used to introduce words of data into a program during assembly. Numbers introduced in this way are often referred to as constants. The pseudo-operation, DUP, causes a sequence of symbolic instructions to be duplicated a specified number of times. DUP is often used with VFD to generate tables of data.

#### OCT — OCTAL DATA

The OCT pseudo-operation is used to create binary data expressed in octal form. It consists of a symbol or blanks in the location field, the operation code OCT in the operation field, and one or more subfields, each containing a signed or unsigned octal integer, in the variable field. The symbol in the location field is the address of the pseudo-operation.

The subfields of the variable field are separated by commas; any number of subfields is permissible, but the last subfield must be terminated by a blank.

The effect of this operation is to convert each subfield to a binary word; these words are assigned to successively higher storage locations as the variable field is processed from left to right. If there is a symbol in the location field, it refers to the first word of data generated.

An example of the data generating function of the OCT pseudo-operation is:

```
OCT  -2345677777, 63, 47, 5,
```

generates data in core locations as follows:

LOCATION	DATA
5000	-2345677777
5001	+0000000063
5002	+0000000047
5003	+0000000005
5004	+0000000000

#### DEC — DECIMAL DATA

The DEC pseudo-operation is used to create words of data expressed as decimal numbers. DEC is identical to OCT, except that the subfields of the variable field are taken to be decimal data items. It consists of a symbol or blanks in the location field, the operation code DEC in the operation field, and one or more subfields, each containing a decimal data item, in the variable field.

The subfields of the variable field are separated by commas; any number of subfields is permissible, but the last subfield must be terminated by a blank.

The effect of this operation is to convert each subfield to a binary word; these words are stored in successively higher storage locations as the variable field is processed from left to right. If there is a symbol in the location field, it refers to the first word of data generated.

An example of the data generating function of the DEC pseudo-operation is:

```
DEC  15, -24, 9, 107, ,
```

LOCATION	DATA
1000	+0000000017
1001	-0000000030
1002	+0000000011
1003	+0000000153
1004	+0000000000
1005	+0000000000

## BCI — BINARY CODED INFORMATION

The BCI pseudo-operation is used to create binary coded character data. Each data word generated by this pseudo-operation consists of six 6-bit characters in standard BCD character code. It consists of a symbol or blank in the location field, the operation code BCI in the operation field, two subfields in the variable field — the count subfield, which consists of an integer followed by a comma (a null expression specifies a count of ten), and the data subfield, whose length is determined by the count subfield.

The number in the count subfield specifies the number of six-character words to be generated; the number of characters in the data subfield is the number in the count subfield multiplied by six. Since the count subfield determines the total length of the variable field, the comments field is assumed to begin immediately following the end of the data subfield, and no blank character is needed to separate the comments field from the variable field.

Thus, the BCI operation introduces data words into consecutive locations, the number of words generated being equal to the number in the count subfield. If there is a symbol in the location field, it refers to the first word of data generated. An example of the data generating function of the BCI pseudo-operation is:

```
BCI 2, BCD MESSAGE COMMENT

LOCATION  DATA
0030  222324604425
0031  626221272560
```

## Storage Allocation Operations

The following operations are used to allocate core storage space:

### BSS — BLOCK STARTING WITH SYMBOL

The operation consists of a symbol or blanks in the location field, the operation code BSS in the operation field, and any expression in the variable field.

The effect of this operation is to reserve a specified amount of storage. This is achieved by increasing the value of the current location counter by the assigned value of the variable field expression. If there is a symbol in the location field, its definition is taken to be the value of the location counter before the increase.

### BES — BLOCK ENDING WITH SYMBOL

This operation functions exactly like BSS, except that a symbol in the location field is defined *after* the location counter is increased.

## BSS AND BES EXAMPLES

An example of reserved storage locations generated with the BSS pseudo-operation is:

```
TRA  START
BSS  4
DEC  97
```

With the TRA instruction located at 1000, the BSS reserves four locations, and the DEC is then located at 1005:

```
1000  TRA  START
1001  BSS  4
1005  DEC  97
```

In a similar fashion, the BES pseudo-operation appearing in the instruction string:

```
TRA  START
BES  4
DEC  97
```

gives the following result:

```
1000  TRA  START
1005  BES  4
1005  DEC  97
```

## Symbol Defining Operations

The following operations are declarative in nature. They are used to define symbols.

### EQU — EQUAL

The operation consists of a symbol in the location field, the operation code EQU in the operation field, and any expression in the variable field.

The effect of this operation is to give the location field symbol the same definition as the variable field expression.

As an example, consider the use of the EQU pseudo-operation in the following instruction string:

```
FSTL  CLA  TMP1
      EQU  *
      ADD  TMP2
```

If the CLA instruction is assigned to location 102, the symbol FSTL would be defined as a symbol, which can be relocated in the program, whose value is 103; the add instruction would then be assigned to location 103. Note that the occurrence of the EQU between two instructions does not alter the sequence of locations assigned by the assembler.

### MAX — MAXIMUM

The MAX pseudo-operation defines the symbol in the location field to have the value of the maximum of the expressions in the variable field. The operation consists of a symbol in the location field, the operation code MAX in the operation field, and a series of expressions, separated by commas, in the variable field.

## MIN — MINIMUM

This pseudo-operation is the opposite of MAX; it uses the expression with the lowest value definition.

## MAX AND MIN EXAMPLE

As an example of MAX and MIN use, assume that records of three different sizes are to be used in a program. The routine is:

LOCATION	OPERATION	ADDRESS, TAG	COMMENT
SIZE1	EQU	xx	
SIZE2	EQU	xx	Addresses to be supplied by user.
SIZE3	EQU	xx	
BUFSZ	MAX	SIZE1, SIZE2, SIZE3	
BUFR	BES	BUFSZ	
ERRSZ	MIN	SIZE1-1, SIZE2-1, SIZE3-1	

The MAX pseudo-operation sets the maximum size of the records and may be referred to by the BES to reserve enough storage space. The MIN may then be used to find records that are too small and are, therefore, errors.

## BOOL — BOOLEAN

The BOOL pseudo-operation consists of a symbol in the location field, the operation code BOOL in the operation field, and an unsigned octal integer in the variable field. The symbol in the location field will be defined as being equal to the integer in the variable field. For example, in the statement:

```
START    BOOL    1200
```

START is defined as being equal to 1200<sub>8</sub>.

## Location Counter Operations

The programmer may use an indefinite number of location counters, represented by symbols of his choice. The operations USE, BEGIN, and ORG control these counters.

## USE

The operation consists of blanks in the location field, the operation code USE in the operation field, and a single symbol or blank in the variable field.

The effect of this operation is to place succeeding cards under control of the location counter represented by the variable field symbol. The location counter in control at the time of the USE is suspended at its current value, and will be continued from this value if reactivated by another USE. If no USE is given, the instruction will be assembled using the location counter represented by blanks.

LOCATION	OPERATION	ADDRESS, TAG
	.....	
	.....	
FINAL	.....	
	USE	A
	BEGIN	A, FINAL + 47
	CLA	

Instructions preceding USE are under machine location counter control. When the USE is encountered, its address (A) is used to set up a different control counter. The BEGIN operation's address of A is modified by FINAL + 47. This means that the location of the CLA instruction is 47 locations after the location of FINAL. Succeeding instructions are located at CLA + 1, CLA + 2, CLA + 3, etc., until location control is changed to another location counter.

## BEGIN

The operation consists of blanks in the location field, the operation code BEGIN in the operation field, and two subfields in the variable field — the first subfield is a location counter symbol, followed by a comma, and the second is any expression. The definition of the expression in the variable field is used as the initial definition for the given location counter.

If no BEGIN is given for the blank location counter, its initial definition is taken to be 0. If no BEGIN is given for the *n*th location counter (considered in location counter order), its initial value is taken to be one more than the last (not necessarily the highest) value reached by the *n*-1st location counter. A BEGIN may appear anywhere in the program (under control of any location counter).

## ORG — ORIGIN

The operation consists of a symbol or blanks in the location field, the operation code ORG in the operation field, and any expression in the variable field.

The ORG operation performs the following functions:

1. The current location counter is reset to the definition of the expression in the variable field.
2. The symbol in the location field, if any, is given this definition. If, in a relocatable assembly, the variable field consists entirely of numbers, the ORG will be taken as absolute. Thus:

```
ORG 5
```

will origin at absolute location 5. To origin at the fifth word from the beginning of the program (that is, for the field to be relocatable), one must write:

```
ORG START + 4
```

where START is the symbol attached to the first program location.

## END

The END operation is used to signal the end of the symbolic deck. It consists of blanks in the location field, the operation code END in the operation field, and a symbolic expression in the variable field. In both absolute and relocatable assemblies, this pseudo-operation terminates (ends) the assembly. The END instruction must be present and must be the last card in the symbolic card deck being assembled.

# Instruction Descriptions and Use

### Instruction Specifications

A symbolic instruction consists of four major divisions: location field, operation field, variable field, and comments field. The location field normally contains a name by which other instructions may refer to the instruction named. The operation field contains the machine operation or pseudo-operation, and the variable field normally contains the location of the operand. The comments field exists for the convenience of the programmer and plays no part in directing the computer.

Symbolic instructions are printed on the coding form (Figure 44) or are punched on a card in the following format (one instruction to a line or to a card):

The *Location Field*, which may be blank, occupies columns 1-6.

*Column 7* is always blank.

The *Operation Field* begins in column 8 and is from three to seven characters long.

A *Blank Column* separates the operation field and the variable field, which may begin in column 12 but may not begin after column 16.

## Symbolic Coding Form

Problem			Date	Page		of	
Coder			Address, Log, Decrement/Count	Comments		Identification	
Location		Operation				72	73
1	2	6	7	8			80

Figure 44. Symbolic Coding Form

The *Variable Field* does not normally extend beyond column 71 and must be followed by a blank column to separate it from the comments field.

The *Comments Field* follows the variable field and extends through column 80. If there is no variable field, the comments field may not begin before column 17.

Columns 73-80 are normally used for identification and serialization.

This section defines the computer instructions. The instruction format, shown in MAP language, appears with each instruction description. Preceding the format is the full name of the instruction. For example, the first instruction described appears:

Clear and Add — CLA      Y,T

This means that the instruction is a clear and add instruction with its operation symbolically expressed as CLA. The number of spaces between the operation part (CLA) and the variable field (Y, T) is four on the coding form. This gives the possible total of seven symbolic characters for the operation code. If the operation code were four characters long, only three spaces would separate the code from the variable field.

The comma (,) symbol in the variable field may be seen in the field heading in Figure 44. The Y symbol is used when the instruction requires an address part. A comma follows the Y symbol if indexing (specified by the T symbol) is to be used with the instruction. A second comma symbol follows the T symbol if decrementing or counting (specified by the V symbol) is a part of the instruction. For example, the variable length multiply instruction format is VLM      Y, T, v. If an instruction does not use indexing but does use the count field, the T symbol is omitted and the instruction format becomes: VLM      Y, , v. *Note that no blank spaces occur in the variable field.*

Instructions using indirect addressing are designated by use of the (\*) asterisk symbol immediately following the operation code. For example, a normal add instruction is expressed as ADD      Y, T; an add instruction using indirect addressing is expressed as ADD\*      Y, T.

Instruction descriptions use special terms and abbreviations:

1. C(Y) denotes the contents of storage location Y, where C(AC), C(MQ), and C(SR) denote the contents of the accumulator, multiplier-quotient, and storage registers. For example, C(MQ)<sub>S, 1-17</sub> is read "the contents of positions S, 1 through 17 of the MQ register." When subscripts are not used, the entire register is implied. For example, C(AC) denotes the contents of accumulator positions S, Q, P, 1-35, inclusive.

2. When a register or part of a register, or a core storage location is cleared, the cleared part is reset to zeros.

3. The negative of a number is the number with its sign position reversed.

4. The magnitude of a number is the number with its sign position considered positive (a zero in position S corresponds to a positive sign).

5. In the alphabetic code of the instruction:

- a. The letter Q designates the MQ register.

- b. The letter X in the second or third position designates use of an index register.

- c. The first letter of all transfer instructions is a T.

### Fixed-Point Arithmetic Instructions

When dealing with fixed-point numbers, the first of the 36 data bits contains the algebraic sign of that number. A 0 signifies a positive number and a 1 signifies a negative number. The remaining 35 positions contain the magnitude of the number. When fixed-point instructions are used, the programmer must decide where the point is to be located. On the computer, the point that separates the integral part from the fraction part is termed a binary point.

Before any arithmetic operations can be executed, one of the numbers involved in the operation must be taken from core storage and placed in the appropriate CPU register. The arithmetic instruction is then given and the second number is brought from core storage and placed in the storage register.

The problem of A + B could be solved with two instructions. The first would clear (or destroy) the formed contents of the accumulator register and place the first number in that register. The second would be brought from core storage, placed in the storage register, and then combined (or added) with the number in the accumulator. The actual adding occurs in the adders. When addition is complete, the result is placed back in the accumulator (which destroys the first number). The format of the first instruction (clear and add) is shown to demonstrate relationship between the symbols and the form.

All succeeding instruction formats are shown with spacing between the operation field and the variable field. The location field is not shown.

Clear and Add — CLA      Y,T

Location				Operation				Address, Tag, Decrement/Count			
1	2	6	7	8	9	10	11	12	13	14	15
				C	L	A					Y, T

The C(AC)<sub>S, 1-35</sub> (contents of the accumulator) are replaced with the C(Y) (contents of the Y storage location). P and Q of the AC are set to zeros and the C(Y) remain unchanged.

### Add — ADD Y,T

The  $C(Y)$  are algebraically added to the  $C(AC)$ . The resulting sum is placed in the  $AC$ . The  $C(Y)$  are unchanged. Numbers of the same magnitude but different signs give a zero result, whose sign is the same as the sign of the original  $AC$ . A carry from position 1 turns on the  $AC$  overflow indicator.

To state the problem in greater detail, assume that factor A is in core storage location 100 while factor B is in location 101. The problem is then written:

```
CLA 100
ADD 101
```

If factor C (held in storage location 102) is added to the problem, the formula becomes  $A + B + C$  and the instructions are increased to:

```
CLA 100
ADD 101
ADD 102
```

After the result is obtained, it should be stored in core storage so that it may be used later in the program for further arithmetic operations or recorded on an output device. A storage location is assigned by the programmer to receive the result (assume location 500). The program instruction used to store the contents of the accumulator register is:

### Store — STO Y,T

The  $C(AC)_{S, 1-35}$  replace the  $C(Y)$  and the  $C(AC)$  remain unchanged. The program is then increased to:

```
CLA 100
ADD 101
ADD 102
STO 500
```

If the number to be placed in the accumulator (before the addition) has the wrong sign, a clear and subtract (CLS) instruction may be used instead of the CLA.

### Clear and Subtract — CLS Y,T

The negative of the  $C(Y)$  replaces the  $C(AC)_{S, 1-35}$ . Positions P and Q of the  $AC$  are set to zero. The  $C(Y)$  are unchanged. The negative of a number is that number with its sign position reversed.

If the difference between the numbers A and B is needed instead of the sum of these numbers, a subtract (SUB) instruction may be used instead of the ADD instruction.

### Subtract — SUB Y,T

The  $C(Y)$  are algebraically subtracted from the  $C(AC)$ . The difference replaces the  $C(AC)$  and the  $C(Y)$  are unchanged. As with the ADD instruction, overflow is possible from position 1 of the  $AC$  to position P and from P to Q, but carries from position Q are lost.

A combination of both arithmetic operations would occur with the formula  $A + B - C$  and store the result at location 500, and would be written:

```
CLA 100
ADD 101
SUB 102
STO 500
```

Figure 45 shows a simplified flow chart for the CLA, CLS, and CAL instructions (discussed later).

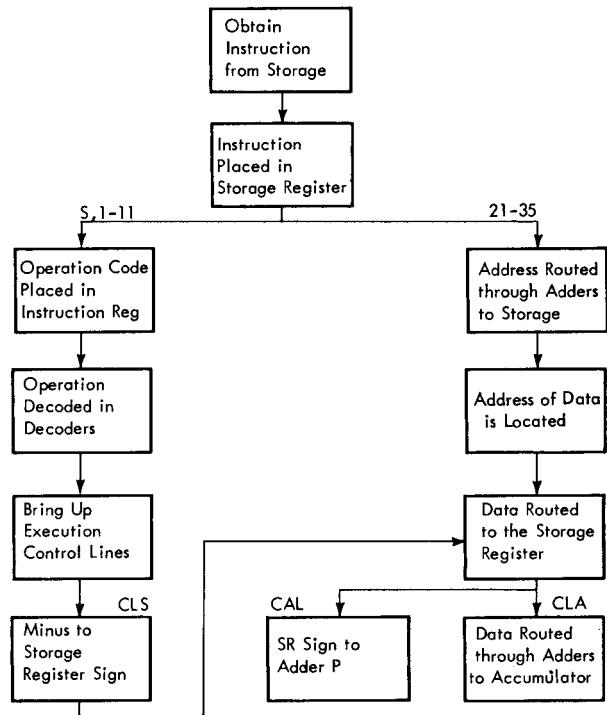


Figure 45. CLA, CLS, and CAL Flow Chart

Figure 46 shows the flow chart for the ADD and SUB instructions. The terms complement and true form are explained in the Complement Arithmetic section.

### Problems

10. Write a program to solve:  $A + B - C + D$  and store the result at location E.
11. Write a program to solve:  $A - B + C - (D - E)$  and store the result at location F.

### Multiply and Divide Operations

The arithmetic operations multiply and divide are accomplished in much the same manner as with add or subtract, except that the multiplier-quotient (MQ) register is used in addition to the accumulator register and the adders.

With a multiply operation, the multiplier factor must be placed in the MQ register before execution of the

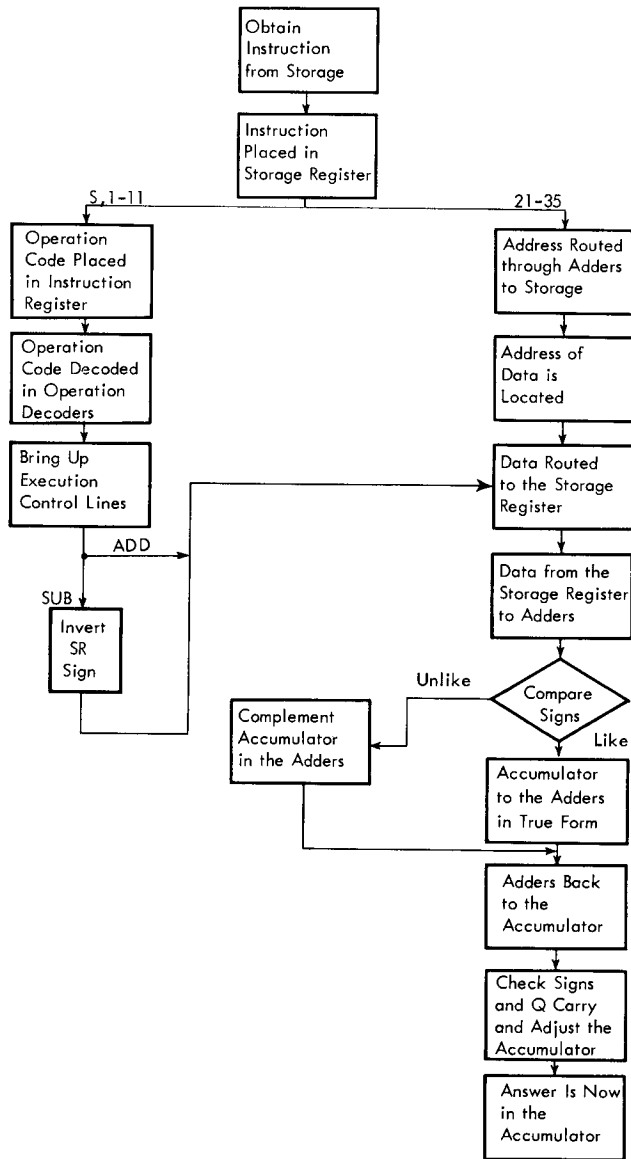


Figure 46. ADD and SUB Flow Chart

actual multiply instruction. This is accomplished with a load multiplier-quotient (LDQ) instruction.

### Load Multiplier-Quotient – LDQ Y,T

The  $C(Y)$  are loaded into the MQ and the  $C(Y)$  are unchanged. After the MQ register is loaded with the multiplier, the multiply instruction may be executed.

### Multiply – MPY Y,T

The  $C(Y)$  are multiplied by the  $C(MQ)$ . The 35 most significant (high order) bits of the 70-bit product replace the  $C(AC)_{1-35}$ , and the least significant (low order) bits replace the  $C(MQ)_{1-35}$ .  $C(AC)_{Q,P}$  positions are cleared to zero. The signs of the AC and MQ are set to the algebraic sign of the product. The number of

bits to the right of the binary point of the first factor added to the number of bits to the right of the binary point of the second factor give the total number of bits to the right of the binary point in the product. The  $C(Y)$  are unchanged.

The programmer must know the size of the product that is possible for his problem. If this product cannot exceed 35 bits, the complete product will be in the MQ at the end of the MPY. In this case, a store multiplier-quotient (STQ) instruction may be used to get the product into core storage.

Figure 47 shows the flow chart for the MPY, VLM, and VMA instructions.

### Store MQ – STQ Y,T

This instruction places the  $C(MQ)$  into the location specified by Y. The  $C(MQ)$  remain unchanged.

For the formula  $A \times B$  and store the product at C, the program may be written:

Location	Operation	Address, Tag, Decrement/Count
1 2		
6 7	LDQ	A
	MPY	B
	STQ	C

If the possibility of a product with more than 35 bits exists, the higher-order bits of the product will end up in the accumulator. The programmer may store the high-order product in one storage location (using the STO instruction) and the low-order product in another location (using the STQ), or he may adjust the entire product with shift and test instructions explained later.

The execution of a multiply instruction occurs as follows and assumes that the MQ register has been loaded with the multiplier.

1. The  $C(Y)$  are tested and, if the magnitude of the  $C(Y)$  is zero, the  $C(AC)$  and  $C(MQ)$  are cleared to zero. In this case, step 2 is skipped and step 3 occurs.

2. If the magnitude of the  $C(Y)$  is not zero, the  $C(AC)_{Q,P,1-35}$  are cleared to zero and the multiplication proceeds.
  - a. If  $MQ_{35}$  contains a 1 bit, the  $C(Y)_{1-35}$  are added to the  $C(AC)$ . The  $C(AC)_{Q,P,1-35}$  and the  $C(MQ)_{1-35}$  are then shifted right one position.
  - b. If  $MQ_{35}$  contains a 0 bit, the  $C(AC)_{Q,P,1-35}$  and  $C(MQ)_{1-35}$  are shifted right one position.

3. If the signs of the MQ and location Y are the same, the signs of the AC and MQ are made positive. If the signs differ, the signs of the AC and MQ are made negative.

As an example, assume that the AC, MQ, and location Y are four bits long instead of 35. The following sequence of steps would occur during a multiplication. The number 15 ( $13_{10}$ ) is in the MQ (multiplier) and



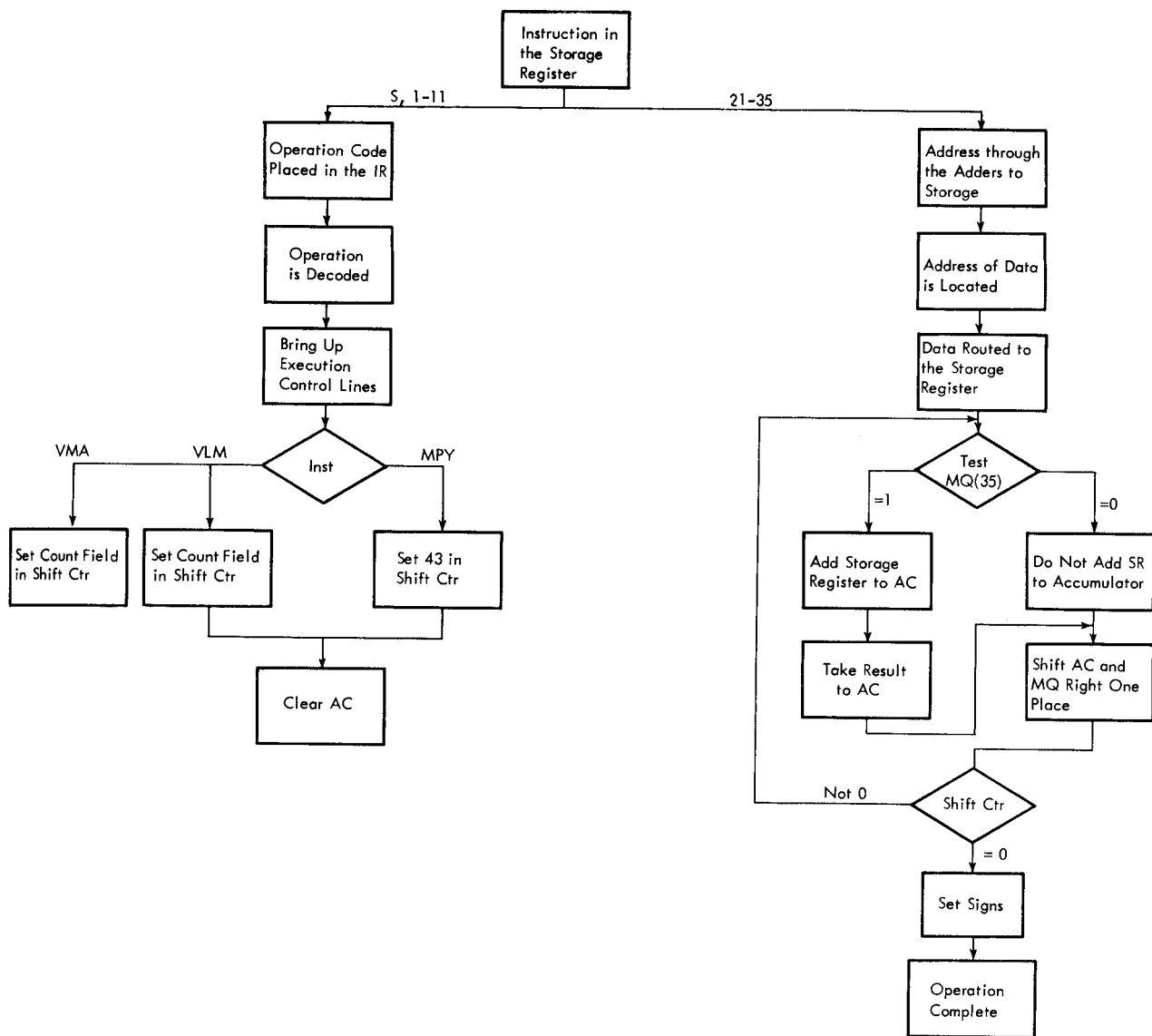


Figure 47. MPY, VLM, and VMA Flow Chart

the  $C(Y)$  are 6 (multiplicand). Figure 48 shows the actual bit configuration in each register (after the step is complete).

### Problems

12. The following quantities are stored in the symbolic storage locations as fields of a pay record. Compute net pay and store the amount in  $PYRCD+4$ . All quantities are assumed to be plus, and results are not larger than 35 bits.

SYMBOLIC LOCATION	FIELD NAME
PYRCD	Employee's Number
PYRCD + 1	Base Pay
PYRCD + 2	Overtime Pay
PYRCD + 3	Deductions
PYRCD + 4	Net Pay (to be computed)

Contents of	Shift	
AC MQ Y	Ctr	Comments
0000 1101 0110	4	Initial contents of the registers. MQ (35) ready to be tested.
0110 1101		C (Y) added to AC since MQ (35) is a 1.
0011 0110	3	C (AC, MQ) shifted right one place. Test MQ (35).
0001 1011		No addition, since MQ (35) contained a 0.
	2	C (AC, MQ) again shifted right one place and MQ (35) is tested.
0111 1011		C (Y) added since MQ (35) is a 1.
0011 1101	1	C (AC, MQ) shifted right and MQ (35) tested.
1001 1101		C (Y) added since MQ (35) is a 1.
0100 1110	0	C (AC, MQ) shifted right. At this point, the shift counter (set initially to a binary 4 in this example) has been reduced to 0 and the process stops with the eight-bit product in the AC and MQ registers. Note: In normal machine operation, the shift counter is set to a binary 43 (which is equal to 35 decimal shifts) automatically.

Figure 48. Multiply Sample Example

13. The following quantities are stored in storage locations as fields of a parts inventory record. Compute stock balance and availability.

SYMBOLIC LOCATION	FIELD NAME
PRTIN	Receipts (+ Sign)
PRTIN + 1	Withdrawals (- Sign)
PRTIN + 2	Adjustments (+ Sign)
PRTIN + 3	On Order (+ Sign)
PRTIN + 4	Reserved for Service (+ Sign)

Stock Balance = receipts - withdrawals + adjustments

Availability = stock balance, + on order, - reserved for service

### Store Zero - STZ Y,T

The store zero instruction may be used to change the contents of an entire core storage location to zeros. The C(Y) are replaced by 0 bits (sign of Y is made plus).

### Variable Length Arithmetic Instructions

Three variable length arithmetic instructions are provided. Positions 12-17 of these instructions designate a count (V) field. It is possible to express a count value up to  $77_8$  with this field. However, counts of  $60_8$  or larger result in placing 1 bits in positions 12 and 13 of the instruction and cause indirect addressing to occur. Counts larger than  $57_8$  should not be used with these instructions.

The contents of the V field are placed in the shift counter instead of the  $43_8$  ( $35_{10}$ ) normally placed there during a multiply or divide instruction. This means that the time required to complete any variable length instruction is a direct result of the V field contents. If the count field of any of these instructions is zero, the instruction is treated as a no-operation and the computer takes the next sequential instruction and proceeds from there.

### Variable Length Multiply - VLM Y,T,V

This instruction multiplies the C(Y) by the V low-order bits of the MQ register to produce a 35 plus V bit product. The 35 most significant bits of the product replace the C(AC)<sub>1-35</sub>, and the least significant bits replace the C(MQ)<sub>1-V</sub>. The C(AC) Q and P positions are set to zero. The remaining 35 minus V positions of the MQ contain the original 35 minus V high-order bits of the MQ.

The signs of the AC and MQ are set to the algebraic sign of the product. If V is zero, the VLM is treated as a no-operation and the computer takes the next sequential instruction and proceeds from there. If V is not zero, but the C(Y) are zero, the C(AC) and C(MQ) are set to zeros. If the MQ and Y signs are the same, the AC and MQ signs are made positive; if the MQ and Y

signs differ, the AC and MQ signs are made negative. If V contains 1 bits in positions 12 and 13, indirect addressing occurs. Counts (V) larger than  $35_{10}$  are meaningless.

### Variable Length Multiply and Accumulate - VMA Y,T,V

This instruction is similar to the VLM instruction except that the C(AC)<sub>Q, P, 1-35</sub> are not cleared before the multiplication begins. Thus, the VMA generates the sum of the magnitude of the C(AC)<sub>Q, P, 1-35</sub> and the magnitude of 35 + V bit product. If AC positions P and Q originally contain 1 bits, a carry may be lost during the accumulation, and the overflow indicator will not be turned on. The V least significant bits of the product replace the C(MQ)<sub>1-V</sub>. The 36 most significant bits replace the C(AC)<sub>P, 1-35</sub>; AC<sub>Q</sub> is set to zero. The remaining 35 minus V positions of the MQ contain the original 35 minus V high-order bits of the MQ. The signs of the AC and MQ are set to the algebraic sign of the product.

If V is initially zero, the instruction is treated as a no-operation and the computer takes the next sequential instruction. If V is not zero, but the C(Y) are zero, the instruction is interpreted as an LRS instruction of V places and the signs of the AC and MQ are set to the sign of the product of the C(Y) and the original C(MQ).

Figure 49 shows register content, both before and after a variable length multiply operation.

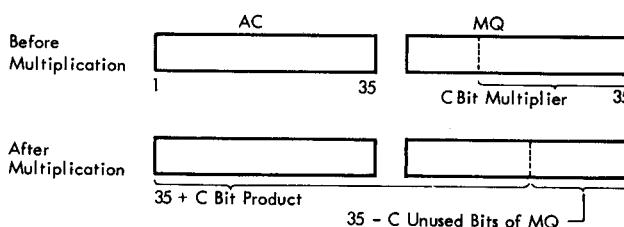


Figure 49. Variable Length Multiply Formats

### Divide or Proceed - DVP Y,T

The divide operation assumes prior loading of the MQ and AC registers with the dividend. Maximum possible dividend is 70 bits. Dividend loading may be accomplished with a LDQ instruction if the dividend is 35 bits or less and it is known that the entire AC is set to zero, or with a CLA and a LDQ if the dividend exceeds 35 bits.

The C(AC)<sub>Q, P, 1-35</sub> and the C(MQ)<sub>1-35</sub> are treated as a 70-bit dividend, plus sign, and the C(Y) as a 35-bit divisor. If the magnitude of C(Y) is greater than the

magnitude of  $C(AC)$ , division takes place. A 35-bit quotient replaces the  $C(MQ)_{1-35}$  and the remainder replaces the  $C(AC)_{1-35}$ . The MQ sign is the algebraic sign of the quotient, and the AC sign is the sign of the dividend. If the magnitude of the  $C(Y)$  is less than or equal to the magnitude of the  $C(AC)$ , division does not take place, the divide check indicator is turned on, and the computer program proceeds to the next sequential instruction. The  $C(Y)$  are unchanged.

Execution of the divide instruction occurs as follows and assumes prior loading of the dividend.

1. The  $C(AC)$  and  $MQ_{1-35}$  are shifted left one position, creating a zero in  $MQ_{35}$ .

2. If the magnitude of the  $C(Y)$  is less than or equal to the magnitude of  $C(AC)$ , the magnitude of  $C(Y)$  is subtracted from the magnitude of  $C(AC)$  and a 1 bit replaces the 0 bit in  $MQ_{35}$ . Step 1 is then repeated.

3. If the magnitude of the  $C(Y)$  is greater than the magnitude of the  $C(AC)$ , the computer returns to step 1.

These steps occur 35 times for each division instruction. As an example, again assume that the computer works with only 4 bits. The problem is then  $66 \div 5$ . In Figure 50, the binary numbers with each step represent the result after the completion of that step.

The programmer must remember the possibility of a remainder after a divide instruction. He may decide to disregard it, check for it and round the quotient if a remainder exists, or use a STO instruction to store the remainder with a STQ to store the quotient.

#### Variable Length Divide or Proceed — VDP Y,T,V

The  $C(AC)_{Q,P,1-35}$  and the  $C(MQ)_{1-C}$  are treated as the dividend plus sign, and the  $C(Y)$  are treated as a 35-bit

Contents of			Shift Ctr	Comments
AC	MQ	Y		
0100	0010	0101	4	Initial contents. C (AC) are less than C (Y); division takes place.
1000	0100		3	C (AC and MQ) shifted left one place; C (AC) greater than C (Y).
0011	0101			C (Y) subtracted from C (AC) and a 1 replaces MQ (35).
0110	1010		2	C (AC, MQ) shifted left one place; C (AC) greater than C (Y).
0001	1011			C (Y) subtracted from C (AC) and a 1 replaces MQ (35).
0011	0110		1	C (AC, MQ) shifted left one place; C (AC) less than C (Y).
0110	1100		0	C (AC, MQ) shifted left one place; C (AC) greater than C (Y).
0001	1101			C (Y) subtracted from C (AC) and a 1 replaces MQ(35). At this point, the shift counter (set to a binary 4 in this case) has been reduced to 0 and the quotient is complete in the MQ, with the remainder in the AC. Note: In normal operation, the shift counter would have been set to a binary 43 so that 35 decimal shifts could occur.

Figure 50. Divide Sample Example

divisor. A V bit quotient replaces the V low order positions of the MQ. The remainder replaces the  $C(AC)_{1-35}$  and the 35 minus V high-order positions of the MQ. Figure 51 shows the flow chart for the DVP and VDP instructions.

#### Problems

14. Write a program to solve:

$$AX^3 + BX^2 + \frac{CX}{D}$$

and store the result in location ANS. All results are assumed to be no more than 35 positions and whole numbers.

15.  $X_1, X_2, X_3, X_4,$  and  $X_5$  are fractions in storage locations X1, X2, X3, X4, and X5. Write a program to solve:

$$\frac{X_1 X_2 X_3}{X_4 X_5}$$

and store the result in location XANS. Assume that  $X_4 > X_1, X_5 > X_2$ , that results do not exceed 35 positions, and that remainders (if any) are to be ignored.

#### Shifting Operations

Shift instructions are used to move the contents of the accumulator and the MQ register either to the right or the left of their original positions. Except for the rotate MQ left instruction, zeros are automatically inserted in the vacated positions of a register. Thus, a shift larger than the bit capacity of the register causes the contents of the register to be replaced by zeros.

When a shift instruction is decoded during the I cycle, the amount of the shift is determined by the contents of bit positions 28-35 of the shift instruction. This provides a maximum shift of  $377_8$  places. Any number larger than  $377_8$  is interpreted as modulo  $400_8$ , which means that, given any shift count, the actual number of positions shifted with the instruction is the remainder after dividing the shift count by  $400_8$ .

When the contents of a register are shifted right, the result is equivalent to dividing the original contents by a power of 2. Likewise, shifting to the left is equivalent to multiplying by a power of 2 (as long as no significant bits are lost).

In the following description of shift instructions, the number of positions to be shifted is specified by positions 28-35.

#### Accumulator Left Shift — ALS Y,T

This instruction causes the  $C(AC)_{Q, P, 1-35}$  to be shifted left the number of places specified by Y. For example, ALS 3 would shift the  $C(AC)$  three places to the left. The

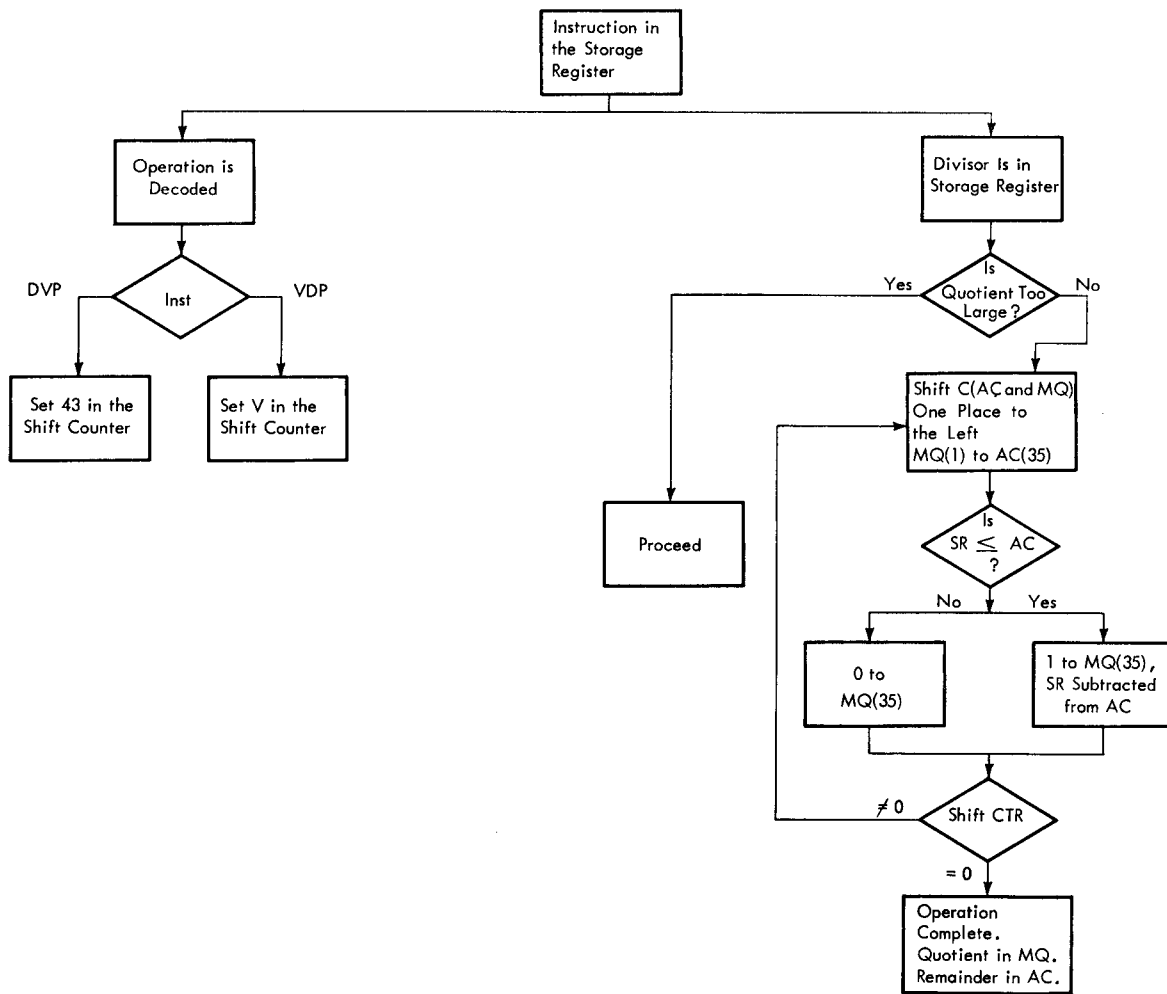


Figure 51. DVP and VDP Flow Chart

sign position of the AC is not shifted (Figure 52). If a 1 bit is shifted into position P from position 1, the AC overflow indicator is turned on. Bits shifted past position Q are lost and vacated positions are filled with zeros.

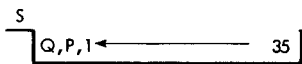


Figure 52. ALS Schematic

### Accumulator Right Shift – ARS Y,T

The  $C(AC)_{Q, P, 1-35}$  are shifted right the number of places specified by Y. The sign position is not shifted (Figure 53), bits shifted from position 35 are lost, and vacated positions are filled with zeros. Bits shifted from position Q enter position P and bits from P enter position 1.

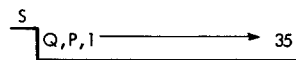


Figure 53. ARS Schematic

### Long Left Shift – LLS Y,T

The  $C(AC)_{Q, P, 1-35}$  and the  $C(MQ)_{1-35}$  are treated as one register. The contents of this register are shifted left the number of places specified by Y. For example, LLS 35 shifts the  $C(MQ)_{1-35}$  to  $AC_{1-35}$ . Bits enter  $AC_{35}$  from  $MQ_1$  (Figure 54). If a 1 bit is shifted into or through position P, the AC overflow indicator is turned on. Bits shifted past position Q are lost, and vacated positions are filled with zeros. The MQ sign position is unchanged and the AC sign is made to agree with it.

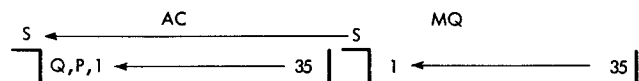


Figure 54. LLS Schematic

**Long Right Shift – LRS Y,T**

The  $C(AC)_{Q, P, 1-35}$  and  $C(MQ)_{1-35}$  are shifted right the number of places specified in Y. Bits enter  $MQ_1$  from  $AC_{35}$  and bits shifted past  $MQ_{35}$  are lost (Figure 55). Vacated positions are filled with zeros. The AC sign is unchanged, and the MQ sign is made to agree with it.

Both the LLS and LRS instructions may be used to move complete words from the MQ to the AC and from the AC to the MQ registers. This results in a reduction of stored instructions. The STQ and CLA instructions could be replaced by an LLS of 35 places, and the LRS could be used instead of the STO and LDQ sequence. LLS

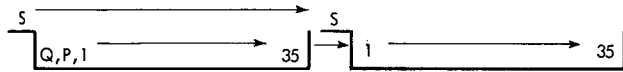


Figure 55. LRS Schematic

or LRS with an address of zero may be used to make AC and MQ signs agree without shifting their data.

**Rotate MQ Left – RQL Y,T**

The  $C(MQ)$  are shifted left the number of places specified by Y. Bits from  $MQ_8$  are routed to  $MQ_{35}$  and from  $MQ_1$  into  $MQ_8$ , in effect, making the MQ register a circular register (Figure 56). For example, RQL 6 takes the six high-order bits (S, 1-5) of the MQ and places them in the low-order six positions (30-35). With the RQL, no bits are lost. Figure 57 is a simplified processing unit flow chart for the ALS, ARS, LLS, LRS, and RQL

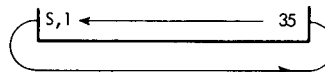


Figure 56. RQL Schematic

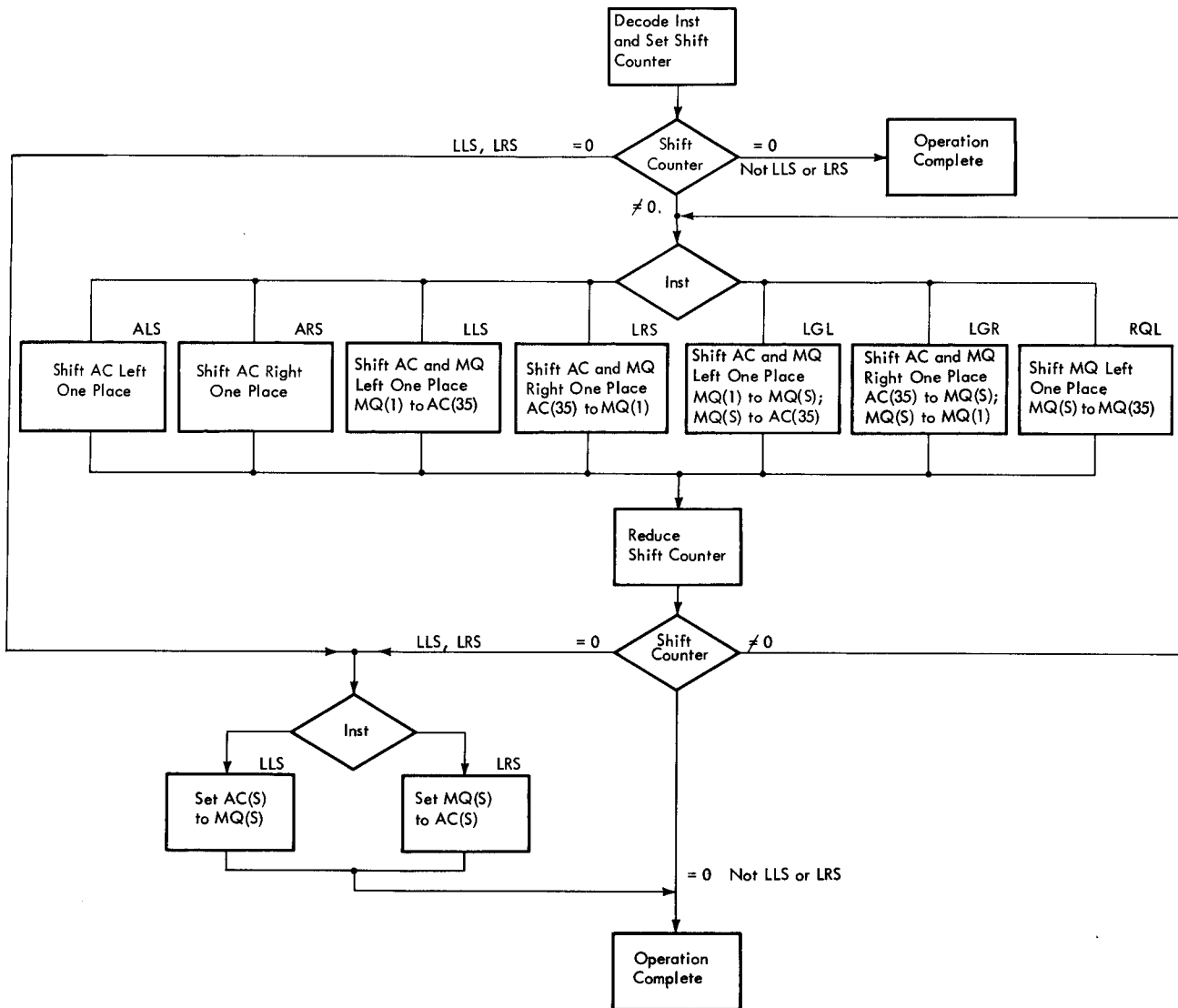


Figure 57. ALS, ARS, LLS, LRS, LGL, LGR, and RQL Flow Chart

instructions. The LGL and LGR instructions shown in this figure are described under "Logical Operations."

### Shifting Problems

Use shift instructions for all multiplication and division.

16. Compute  $2(A + B + C)$  and store result at location P.

17. Compute  $\frac{A + B - C}{16}$  and store result at P.

18. Compute  $(A - B) \times 8$  and store result at P.

### Control Instructions

Instructions that govern the flow of a program and, in particular, those that cause an alteration in the computer's normal process of taking its instructions from sequential core storage locations are called control instructions.

Unconditional transfer instructions specify the location Y from which the computer is to take the next instruction. Conditional transfer instructions also specify a location Y; whether the computer takes its next instruction from location Y or the next sequential location depends on the outcome of a test of some kind. This test is specified by the operation code of the instruction.

Test instructions are similar to conditional control instructions in that they cause some test to be performed. Unlike conditional transfer instructions, however, test instructions do not specify a location Y to which control may be transferred. Instead, the alternative location to which control may be transferred is fixed relative to the location of the test instruction.

### Halt and Proceed — HPR

This instruction causes the computer to halt. The instruction counter contains the location of the next sequential instruction and is displayed on the operator's console. Positions 21-35 (Y), not used by the instruction but displayed in the storage register lights, may be used to identify each particular HPR. This is done by placing an identifying number in positions 21-35 of the HPR.

### Divide Check Test — DCT ,T

If the divide check indicator is on, it is turned off and the computer takes the next sequential instruction. If the indicator is off, the computer skips the next instruction and proceeds from there.

NOTE: The DCT, LBT, and PBT instructions use the address field (Y) for special purposes and no address may be specified. If the T field is used, the operation code itself may be changed (See "Appendix, Instruction List with Formats").

### Low-Order Bit Test — LBT ,T

If the  $C(AC)_{35}$  is a 1 bit, the computer skips the next instruction and proceeds from there. If the  $C(AC)_{35}$  is a 0 bit, the computer takes the next sequential instruction. This instruction may be used to test for an odd or even accumulator.

### P Bit Test — PBT ,T

If the  $C(AC)_P$  is a 1 bit, the computer skips the next instruction and proceeds from there. If the  $C(AC)_P$  is a 0 bit, the computer takes the next sequential instruction. This instruction may be used to test for AC overflow. If the AC overflow indicator is on, it is not turned off by execution of the PBT instruction. Figure 58 shows the flow chart for the PBT, LBT, and DCT instructions.

### Sense Switch Test — SWT Y,T

This instruction tests the status of the sense switch (on the operator's console) specified by Y. If the corresponding switch is down (ON), the computer skips the next sequential instruction and proceeds from there. If the switch is up (OFF), the computer takes the next sequential instruction. For example,  $SWT_2$  tests the status of sense switch 2. There are six switches on the operator's console that may be tested by the SWT instruction.

### Execute — XEC Y,T

This instruction causes the computer to execute the instruction at location Y. Since the location counter is not altered (when Y contains any instruction except

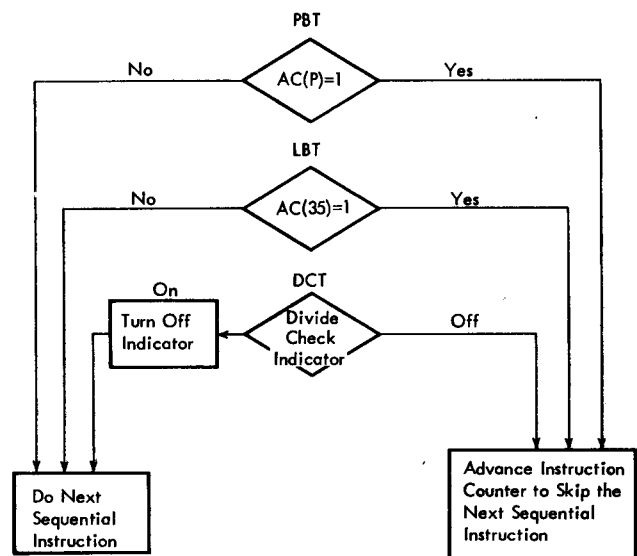


Figure 58. PBT, LBT, and DCT Flow Chart

a successful transfer or test instruction), the program advances to the next sequential instruction following the execute instruction after performing the instruction at location Y. If location Y contains a transfer instruction, it will be executed and program control is altered from the sequential process. If location Y contains a test instruction, the instruction following the execute instruction will be located relative to the execute instruction rather than to the test instruction. Thus, any instruction that changes the instruction counter alters program control when that instruction is executed by the XEC instruction.

**Transfer on No Zero – TNZ Y,T**

If the  $C(AC)_{Q,P,1-35}$  are not zero, the computer takes its next instruction from the location specified by Y and proceeds from there. If they are zero, the next sequential instruction is taken.

Figure 59 shows the flow chart for the TNZ, TPL, TOV, TZE, and TMI instructions.

**Transfer on Plus – TPL Y,T**

If the sign position of the AC is a zero, the computer takes its next instruction from the location specified by Y and proceeds from there. If the sign position is a one, the computer takes the next sequential instruction.

**Transfer on Overflow – TOV Y,T**

If the AC overflow indicator is on, it is turned off and the computer takes its next instruction from the location specified by Y. If the indicator is off, the computer takes the next sequential instruction. Note also that the PBT instruction may be used as an overflow test instruction.

**Transfer – TRA Y,T**

This instruction causes the computer to take its next instruction from the location specified by Y and proceed from there.

**Transfer on Zero – TZE Y,T**

If the  $C(AC)_{Q,P,1-35}$  are zero, the computer takes its next instruction from the location specified by Y. If they are not zero, the computer takes the next sequential instruction.

**Transfer on Minus – TMI Y,T**

If the sign position of the AC is negative (1 bit), the computer takes its next instruction from the location specified by Y and proceeds from there. If the sign position is positive (0 bit), the computer takes the next sequential instruction.

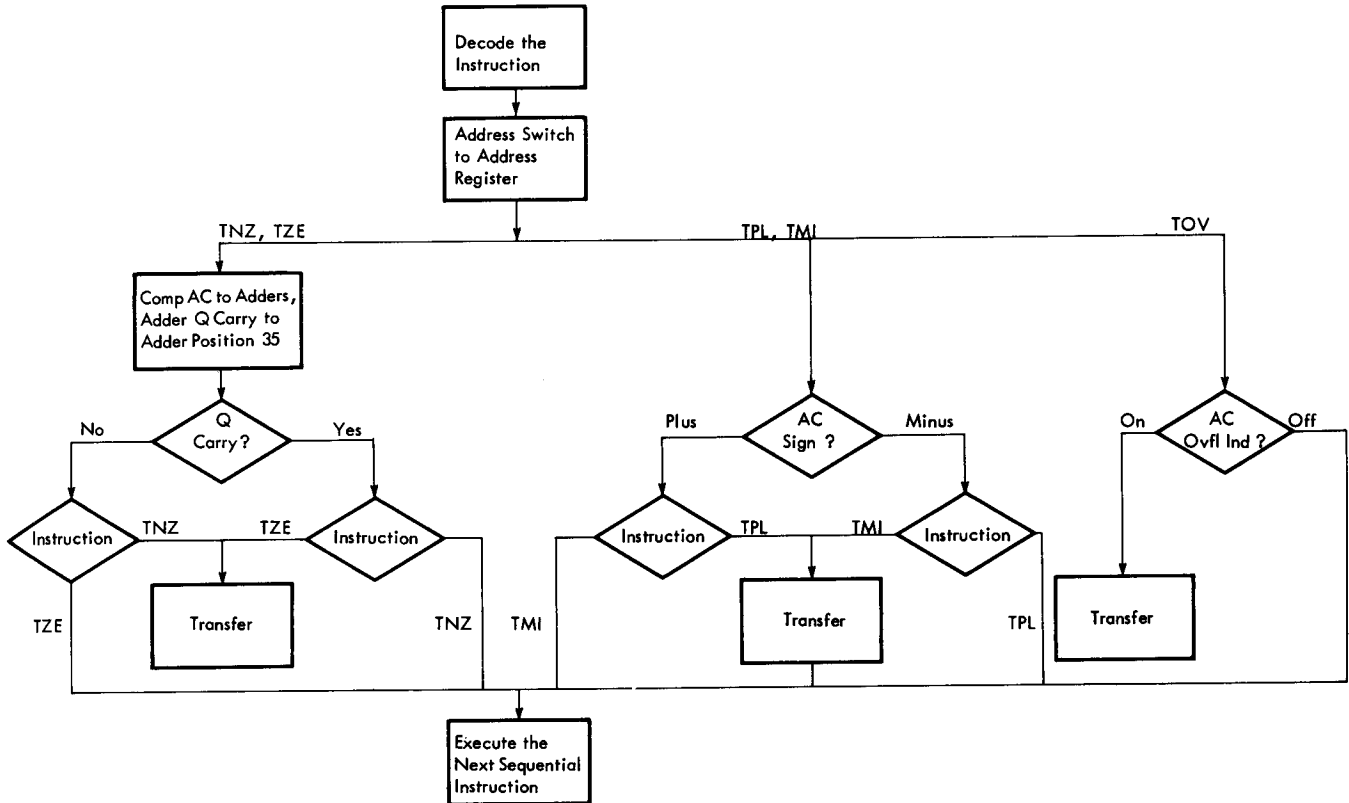


Figure 59. TNZ, TPL, TOV, TZE, and TMI Flow Chart

### Compare Accumulator with Storage — CAS Y,T

If the  $C(AC)$  are algebraically greater than the  $C(Y)$ , the computer takes the next sequential instruction. If the  $C(AC)$  are algebraically equal to the  $C(Y)$ , the computer skips the next instruction and proceeds from there. If the  $C(AC)$  are algebraically less than the  $C(Y)$ , the computer skips the next two instructions and proceeds from there. A plus zero is considered greater than a minus zero. NOTE: The comparison is made on all positions of the AC (including positions P and Q) and the contents of location Y.

### Decrement Field

Some instructions use the decrement part (Figure 60) of themselves or the decrement part of a register or core location in their execution. With some instructions, a portion of the decrement field (positions 15-17) is used as a part of the operation field. Another group of instructions is used to test or alter the contents of index registers. The number or value used to test or alter an index register is contained in positions 3-17 of these instructions.

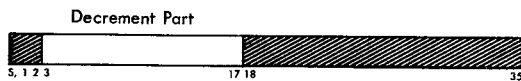


Figure 60. Decrement Field in a Word

### Set Sign Plus — SSP ,T

The sign of the AC is set plus (0 bit). Since the address part of the SSP instruction is a part of the operation code, address modification may change the operation.

### Change Sign — CHS ,T

If the AC sign is plus, it is made minus; if minus, it is made plus. Since the address part of the CHS instruction is a part of the operation code, address modification may change the operation.

### Make Storage Sign Minus — MSM Y,T

The sign position of the location specified by Y is replaced with a 1 bit (made minus). The remainder of the location specified by Y is unchanged. The decrement part of the MSM instruction is a part of the operation code.

### Make Storage Sign Plus — MSP Y,T

The sign position of the location specified by Y is replaced with a 0 bit (made plus). The remainder of the location specified by Y is unchanged. The decrement part of the MSP instruction is part of the operation code.

NOTE: The SSP and CHS instructions are not exactly control instructions but are normally used with or after control instructions. These instructions use the address field for special purposes and no address may be specified. If the T field is used, the operation code itself may be changed (See "Appendix, Instruction List with Formats").

### Storage Minus Test — MIT Y,T

If the sign position of the location specified by Y is minus, the computer skips the next instruction and proceeds from there. If the sign position is plus, the computer takes the next sequential instruction. The decrement part of the MIT instruction is a part of the operation code.

### Storage Plus Test — PLT Y,T

If the sign position of the location specified by Y is plus, the computer skips the next instruction and proceeds from there; if the sign position is minus, the computer takes the next sequential instruction. The decrement part of the PLT instruction is a part of the operation code.

### Enter Keys — ENK ,T

This instruction places the contents of the console entry keys into the MQ register. Since the address part of the ENK instruction is a part of the operation code, address modification may change the operation. A depressed switch is interpreted as the not-zero or ON condition.

### Problems

19. Three numbers are stored in symbolic locations A, B, and C. Determine which is the lowest number and store this number in location LOW (none of the numbers are equal).

20. The fields of an inventory parts record are arranged in storage as follows:

LOCATION	FIELD NAME
PARTN	Part Number
PARTL	Part Location
UCOST	Unit Cost
FIND	Master Part Number

Compare the master part number against the part number of the given record and:

- If master is higher than the given record, transfer to QUIT.
- If master is equal to the given record, transfer to PROCS.
- If master is lower than the given record, transfer to MORE.

21. There are four numbers in locations A, B, C, and D. Program the following:

- Add the four numbers and check for overflow on each addition; if an overflow occurs, keep a count in location OVFLW adding 1 for each overflow.
- Take the sum generated in step a and shift it to the right until a 1 bit appears in  $AC_{35}$ .
- Take the result of step b and test for a one in  $AC_P$ . If there is a 1 bit, shift right one position and store a 1 bit in location PBIT.

NOTE: Other control instructions than those covered in this section exist on the 7040/7044 systems, but they are more concerned with other features of the system and are described with their own feature.



## Indexing Operations

Several techniques may be used to increase program efficiency. One technique is address modification; another is indirect addressing. Two approaches to address modification are considered here: the destructive type and the indexing type.

### Destructive Address Modification

The term destructive means that the original address of the instruction being modified is destroyed as it is modified. Regardless of the computer used, this is the method of address modification used unless the computer is equipped with index registers and indexing instructions. If an application required an instruction to be repeated many times, that instruction would have to be duplicated in the program and stored in core storage. For example, if the contents of 50 word locations were to be added together, 50 add instructions would have to be placed in the stored program. Each add instruction would have, as its address part, the storage location for one of the 50 words.

The technique of modifying an instruction's address may be used to reduce the number of stored instructions. This technique, however, does increase over-all execution time for the problem. Using the same example as above, assume that the 50 word locations are designated *FIRST*, *FIRST + 1*, etc. Figure 61 shows a program that could add the contents of these locations.

Note the use of the \* (asterisk) symbol in the address part. When used this way, the \* means the location of the instruction itself. Thus, the *CLA \*-2* means to bring into the accumulator the contents of the location that is two locations in front of the *CLA* instruction location (*ADD FIRST+1*).

## Indirect Addressing

The concept of address modification may be extended for a large group of instructions for which indirect addressing is provided. This is accomplished by using the V field of the instruction. Positions 12 and 13, when they contain 1 bits, signal indirect addressing.

When indirect addressing is specified, the instruction is executed as follows. Instead of using the address part of the instruction to designate the storage location to be used in the operation, the address part of the addressed location tells the program which storage location is to be used. For example, assume that the address part of location 54 contains 273. If the instruction *ADD 54* (with indirect addressing specified) is executed, the contents of location 273 would be added into the accumulator.

Indirect addressing is specified in symbolic language by placing an \* (asterisk) after the last character of the operation field. Thus, the *ADD 54* instruction, when specifying indirect addressing would be expressed as *ADD\* 54*. Figure 62 shows a sample program using indirect addressing.

The contents of the instruction counter are stored in location 12. Location 13 is designated as symbolic location *BTRAP* in Figure 62. After execution of the channel

Location	Operation	Address, Tag	Comments
		.....	
<i>BTRAP</i>		.....	Store instruction counter at 12 and get next instruction from location 13.
		.....	
	<i>RCT</i>		Restore channel B.
	<i>TRA*</i>	12	Go to 12 for address of next instruction.

Figure 62. Indirect Addressing Example

Location	Operation	Address, Tag, Decrement/Count	Comments	Identification
1 2				72 73 80
	<i>ORG</i>		Locate the program.	
<i>START</i>	<i>CLA</i>	<i>FIRST</i>	Location of first number.	
	<i>ADD</i>	<i>FIRST+1</i>	The address of this instruction is changed.	
	<i>STO</i>	<i>TEMP</i>	Temporary storage location.	
	<i>CLA</i>	<i>*-2</i>	Bring instruction into accumulator.	
	<i>ADD</i>	<i>= 1</i>	Literal.	
	<i>STO</i>	<i>*-4</i>	Store altered instruction.	
	<i>SUB</i>	<i>COUNT</i>	Reduce the number counter.	
	<i>TZE</i>	<i>END</i>	Test for program end.	
	<i>CLA</i>	<i>TEMP</i>	Temporary storage location.	
	<i>TRA</i>	<i>START+1</i>	Return to add next number.	
<i>COUNT</i>	<i>ADD</i>	<i>FIRST+49</i>	Constant for number counter.	
	<i>HPR</i>		Stop.	
<i>FIRST</i>	<i>BSS</i>	<i>50</i>	Reserved storage area.	
	<i>END</i>		Last symbolic instruction.	

Figure 61. Address Modification Example (Destructive)

B trap routine, a restore channel traps (RCT) instruction is executed to restore the channel indicators to a non-trap state. An indirectly addressed TRA\* 12 will then transfer program control to the location contained in the address part of location 12.

### Indexing Concept

Indexing instructions are available as a part of the optional extended performance instruction set. These instructions may be used to modify addresses of existing instructions, reducing the number of core storage locations used for instruction storage.

Indexing is the ability of a computer to combine the contents of an index register with the address portion of an instruction before the instruction is executed. There are two main reasons for indexing: (1) the instruction as it appears in core storage is never changed and therefore its address never has to be initialized (set at the beginning of the program run), and (2) many addresses can be modified by the same index register's contents.

The index registers may be loaded with either true or complement numbers. When combined with the address part of an instruction, the address may be either increased or decreased depending on the type of number (true or complement) the index register started with.

The 7040/7044 systems have three index registers. These registers are termed A, B, and C or 1, 2, and 4. The latter terminology is more convenient for the programmer working in machine language because the numbers 1, 2, and 4 are the octal representation of the addresses of the registers. Index register addresses are specified in a part of the instruction word known as the tag field. The tag field tells the computer whether an instruction is going to use an index register and, if so, which register is to be used. The tag field is located in bit-positions 18, 19, and 20 of the instruction word (Figure 63). By having more than one tag bit in the tag field, two or more index registers may be used by a single instruction. Thus, the contents of the index registers would be combined and the resultant OR (see "Packing and Unpacking") would be used. With some indexing instructions, the omission of 1 bits in the tag field simulates an index register of all zeros.

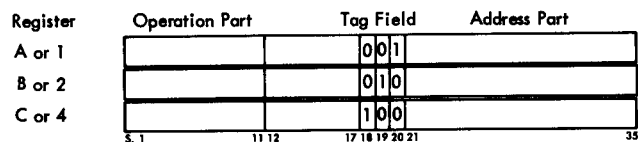


Figure 63. Index Register Tag Bister

The 7040/7044 index registers are 15 positions long — large enough to hold the largest possible storage address. Instructions are available to test index register contents and control program instruction execution depending on that content. The contents of index registers may also be reduced or increased by variable amounts.

### Complement Arithmetic

When index registers are used for address modification, the contents of an index register is always subtracted from an instruction's address. Since neither the address of the instruction nor the contents of the index register is associated with any algebraic sign, it is not possible to accomplish effective address modification by addition in any direct manner. However, addition is accomplished by using complement arithmetic. The following definitions apply to this type of arithmetic:

*The 1's Complement* of a binary number is the number that results by replacing each 1 in a number with a 0 and each 0 with a 1. For example, given the binary number of 101; the 1's complement would be 010. Also, the sum of a binary number and its 1's complement is a binary number composed of all ones (101 + 010 = 111).

*The 2's Complement* of a binary number is defined as the 1's complement of a number increased by one. Thus, for the preceding example, the 2's complement of a number (101) would be 011. If the 2's complement of a number occupies an index register and is used to modify an address, the effective address is the sum of index register contents and the address portion of the instruction. If the true number occupies the index register, the effective address is the difference between index register contents and the address part of the instruction.

Note that since both the contents of an index register and an instruction address are 15-bit numbers, all carries out of the leftmost position are lost.

As an example of the arithmetic involved when index registers are used, assume that index register (XR) 1 contains the binary number 2 and that an ADD instruction with a tag of 1 and an address of 200<sub>8</sub> is to be executed (Figure 64). When the ADD instruction is decoded, the tag bit in position 20 specifies XR1. The contents of XR1 are complemented (2's complement) and placed in the address. Note that index register contents are *always* automatically (2's) complemented when taken to the address. This feature results in subtracting the contents of the XR from the address part of the instruction. The address part of the ADD instruction is also placed in the address; after adding the two numbers, the result (called the effective address) is used in execution of the ADD instruction instead of the

actual address. In this case, the effective address is  $176_8$ .

If the programmer wishes to increase the effective address, the number placed in the XR is inserted in 2's complement form by instruction. Thus, when the address of the instruction and XR contents are combined, the result is an additive process. Using the same facts (as in Figure 64) with the XR contents in 2's complement form, the effective address is now  $202_8$  instead of  $176_8$  (Figure 65).

### Multiple Tags

As previously stated, an instruction may refer to more than one index register by placing multiple 1 bits in the tag field (Figure 66). Thus, a tag of  $3_8$  specifies index registers 1 and 2. Care must be exercised when multiple tags are used. The use of multiple tags results in a "logical or" (see "Packing and Unpacking") of the contents of the specified index registers. For example, if a tag of 3 is given, the 15 positions of index register 1 are matched against the corresponding 15 positions of index register 2. If corresponding positions of each register contain 1 bits, the resultant logical sum is a 1 bit. If both positions are 0 bits the logical sum for that position is a 0 bit.

Assume that index register 1 contains  $03204_8$  (000 011 010 000 100) and index register 2 contains  $03061_8$  (000 011 000 110 001). The instruction ADD  $06521_8$ , with an index tag field of 3, causes the "inclusive OR" (see "Packing and Unpacking") of the contents of the two registers as shown in Figure 67.

Tag Field		Index Registers Specified	
Binary	Octal	Octal	Alpha
000	0	None	None
001	1	1	A
010	2	2	B
011	3	1 and 2	A and B
100	4	4	C
101	5	1 and 4	A and C
110	6	2 and 4	B and C
111	7	1, 2, and 4	A, B, and C

Figure 66. Multiple Tags

The effective address received from the subtraction is  $03234_8$ , which the ADD instruction uses.

### Partial Store Instructions

Two store type instructions, STA and STD, are available which store only parts of a word instead of the whole word. With both of these instructions, the check bit (position 36) of the word stored is automatically changed if necessary.

Index Register 1 Contents	000 011 010 000 100
Index Register 2 Contents	000 011 000 110 001
Inclusive OR'ed Result	000 011 010 110 101 or
	$03265_8$

Figure 67. Inclusive or Example

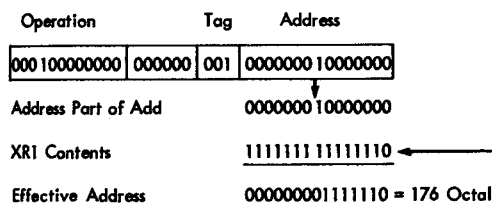


Figure 64. Index Register Arithmetic, Subtracting

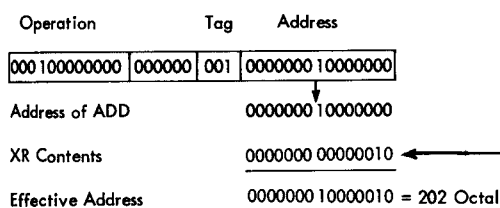
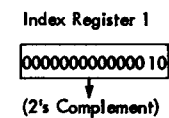
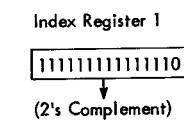


Figure 65. Index Register Arithmetic, Adding



111111111111110



000000000000010

**Store Address — STA Y,T**

The  $C(AC)_{21-35}$  replace the  $C(Y)_{21-35}$ . The  $C(Y)_{8,1-20}$  and the  $C(AC)$  remain unchanged.

**Store Decrement — STD Y,T**

The  $C(AC)_{3-17}$  (decrement part) replace the  $C(Y)_{3-17}$ . The  $C(Y)_{8,1,2,18-35}$  and the  $C(AC)$  remain unchanged.

**Index Register Servicing and Testing**

Computer instructions, when tagged, are subjected to address modification; exceptions are instructions that load, store, modify, or test the contents of an index register. These instructions use the tag field to specify the index registers affected. The following instructions are used for index register (XR) servicing and testing.

**Address to Index True — AXT Y,T**

The value specified in the Y portion of this instruction replaces the contents of the index register specified by the T portion of this instruction. For example, AXT 30,1 places the decimal value 30 (coded in binary format) in index register 1. The instruction itself remains unchanged. A tag of zero results in a no-operation.

**Load Complement of Address in Index — LAC Y,T**

The 2's complement of the  $C(Y)_{21-35}$  replaces the contents of the specified XR. For example, LAC 5,2 takes positions 21-35 of core location 5 and places the 2's complement of this value in index register 2. The  $C(Y)$  remain unchanged. A tag of zero results in a no-operation.

**Load Complement of Decrement in Index — LDC Y,T**

The 2's complement of the  $C(Y)_{3-17}$  replaces the contents of the specified XR. The  $C(Y)$  are unchanged. A tag of zero results in a no-operation.

**Load Index from Address — LXA Y,T**

The  $C(Y)_{21-35}$  replace the contents of the specified XR. The  $C(Y)$  are unchanged. A tag of zero results in a no-operation.

**Load Index from Decrement — LXD Y,T**

The  $C(Y)_{3-17}$  replace the contents of the specified index register. The  $C(Y)$  are unchanged. A tag of zero results in a no-operation. Figure 68 is the flow chart for the LAC, LDC, LXA, and LXD instructions.

**Place Complement of Address in Index — PAC ,T**

The 2's complement of the  $C(AC)_{21-35}$  replace the contents of the specified XR. The  $C(AC)$  are unchanged. A tag of zero results in a no-operation.

**Place Address in Index — PAX ,T**

The  $C(AC)_{21-35}$  replace the contents of the specified XR. The  $C(AC)$  are unchanged. A tag of zero results in a no-operation.

**Place Complement of Decrement in Index — PDC ,T**

The 2's complement of the  $C(AC)_{3-17}$  replace the contents of the specified XR. The  $C(AC)$  are unchanged. A tag of zero results in a no-operation.

**Place Decrement in Index — PDX ,T**

The  $C(AC)_{3-17}$  replace the contents of the specified XR. The  $C(AC)$  are unchanged. A tag of zero results in a no-operation. The flow chart for the PAC, PAX, PDC, and PDX instructions is shown in Figure 69.

**Place Index in Address — PXA ,T**

The entire accumulator is cleared to zero, and the contents of the specified XR are placed in  $AC_{21-35}$ . With a tag of zero, the  $C(AC)$  are set to zero. The  $C(XR)$  are unchanged.

**Place Index in Decrement — PXD ,T**

The entire accumulator is cleared and the contents of the specified XR are placed in  $AC_{3-17}$ . With a tag of zero,

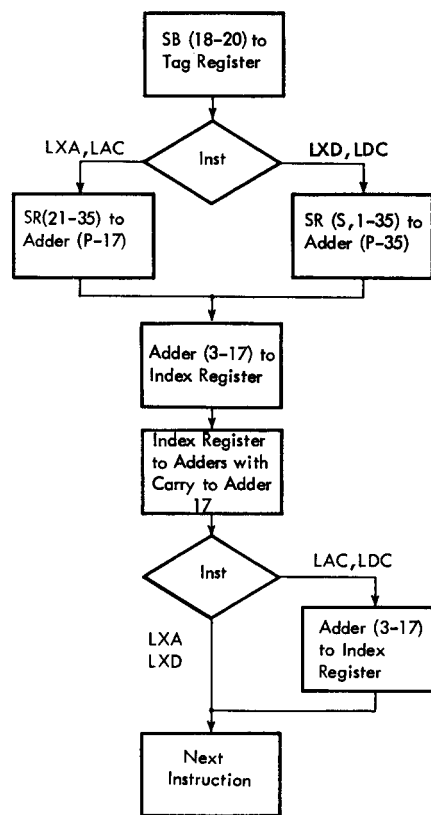


Figure 68. LAC, LDC, LXA, and LXD Flow Chart

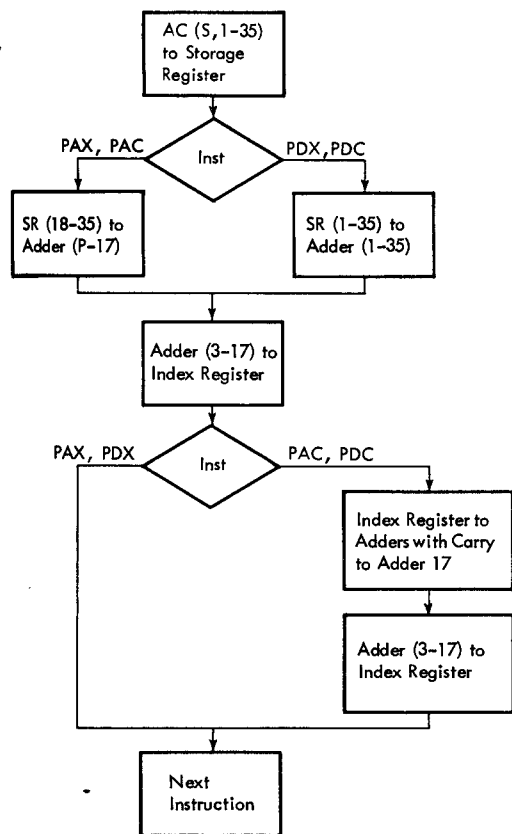


Figure 69. PAC, PAX, PDC, and PDX Flow Charts

the  $C(AC)$  are set to zero. The  $C(XR)$  are unchanged. Figure 70 shows the flow chart for the PXA and PDX instructions.

### Store Index in Address — SXA Y,T

Positions 21-35 of the location specified by Y are replaced by the contents of the specified XR. The  $C(Y)_{8,1,20}$  and the  $C(XR)$  are unchanged. With a tag of zero, the  $C(Y)_{21-35}$  are set to zero.

### Store Index in Decrement — SXD Y,T

The  $C(Y)_{3-17}$  are replaced by the contents of the specified XR. The  $C(Y)_{8,1,2,18-35}$  and the  $C(XR)$  are unchanged. With a tag of zero, the decrement (positions 3-17 of the specified Y) is replaced with zeros. Figure 71 is the flow chart for the SXA and SXD instructions.

### Transfer on Index — TIX Y,T,V

If the  $C(XR)$  specified by T are greater than the value specified by V, the contents of the XR are reduced by

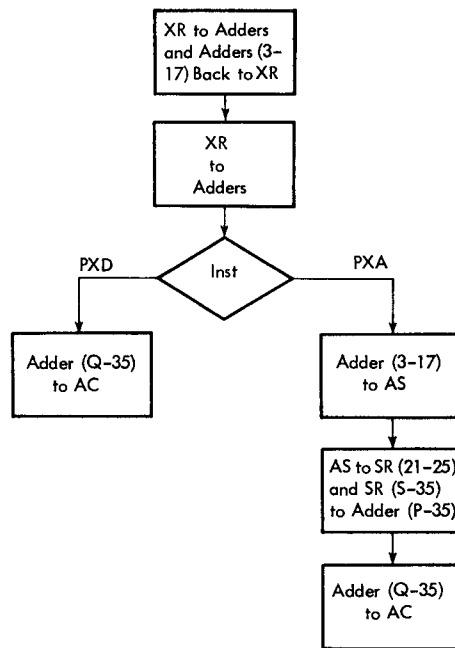


Figure 70. PXA and PDX Flow Chart

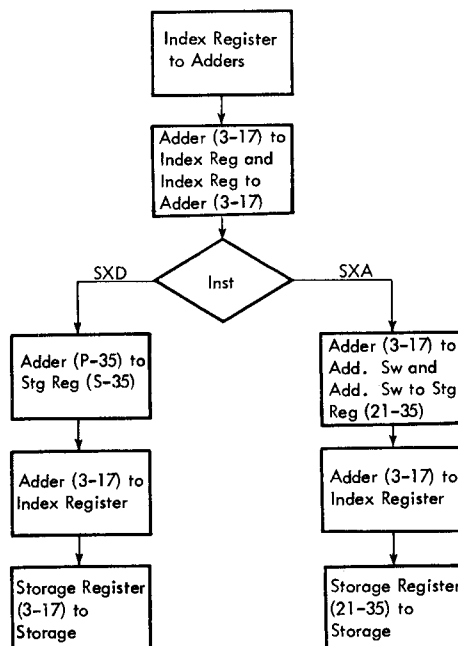


Figure 71. SXA and SXD Flow Chart

V and the computer takes its next instruction from Y. If the  $C(XR)$  are less than or equal to V, the  $C(XR)$  are unchanged and the computer takes the next sequential instruction. With a tag of zero, no transfer occurs.

As an example of the use of the TIX instruction, assume that 50 words are to be added into the accumulator and that the result is to be stored in location

TOTAL. The words are located in the symbolic locations WORD. Figure 72 shows two possible instruction sequences.

**Transfer on No Index – TNX Y,T,V**

If the C(XR) specified by T are greater than V, the C(XR) are reduced by V, and the computer takes the next sequential instruction. When the C(XR) are equal to or less than V, no reduction is made but the computer transfers to location Y. With a tag of zero, a transfer occurs.

**Transfer on Index High – TXH Y,T,V**

If the C(XR) specified by T are greater than V, the computer takes its next instruction from location Y. If the C(XR) are less than or equal to V, the computer takes the next sequential instruction. With a tag of zero, no transfer occurs.

**Transfer with Index Incremented – TXI Y,T,V**

V is added to the C(XR) specified by T. The computer then takes its next instruction from location Y. With a tag of zero, only the transfer occurs.

**Transfer on Index Low or Equal – TXL Y,T,V**

If the C(XR) specified by T are less than or equal to V, the computer takes its next instruction from location Y.

If the C(XR) are greater than V, the computer takes the next sequential instruction. With a tag of zero, a transfer occurs. Figure 73 summarizes the transfer, test, and modify actions of indexing instructions and gives the conditions on each instruction.

Figure 74 shows data flow between storage, accumulator, and index registers (for index transmission instructions). Both true and complement lines are shown with appropriate instructions.

**Indexing Problems:**

22. There are 24 numbers stored in locations N to N+23. Compute and place the sum of the numbers that are positive in location PSUM. The sum will not exceed 35 bits.

23. There are 31 numbers stored in locations M to M+30. Compute and place the sum of the numbers

Actions	Conditions	
	If $XR > V$	If $XR \leq V$
Test and Modify TIX TNX	$C(XR)=XR-V$ and transfer to Y $C(XR)=XR-V$ and take next instruction	Take next instruction Transfer to Y
Test Only TXL TXH	Take next instruction Transfer to Y	Transfer to Y Take next instruction
Modify Only TXI	$C(XR) = XR + V$ and Transfer to Y	

Figure 73. Index Transfer Instruction Summary

Location	Operation	Address, Tag, Decrement/Count	Comments	Identification
WORD	BES	50	Reserve storage locations.	72, 73, 80
	AXI	49, 1	Put 49 into XR 1.	
	CLA	WORD-50	Put first word in accumulator.	
START	ADD	WORD, 1	Get next word.	
	TIX	START, 1, 1	Check for equal XR and V; if unequal, reduce XR by V and transfer.	
	STO	TOTAL	When equal, store result.	
Another program variation could be:				
	AXI	50, 1	Put 50 into XR 1.	
	PID		Clear accumulator.	
START	ADD	WORD, 1	Put first word into accumulator.	
	TIX	START, 1, 1	Check for equal XR and V; if unequal reduce XR by V and transfer.	
	STO	TOTAL	When equal, store result.	
	HPR		Stop.	
WORD	BES	50	Reserve storage locations.	
	END		End of symbolic instructions.	

Figure 72. TIX Instruction Uses

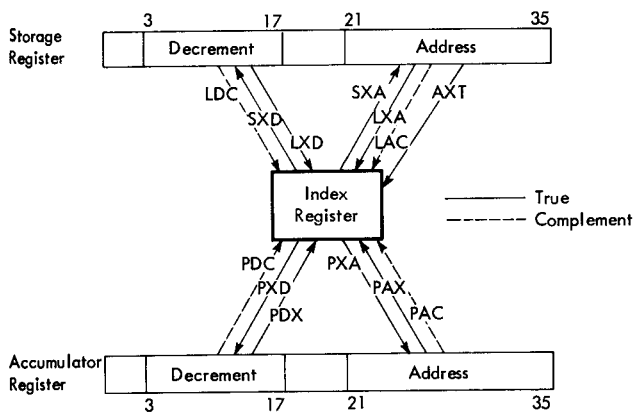


Figure 74. Index Transmission Data Flow

that are positive in location PSUM and the sum of the numbers that are negative in location MSUM.

24. One hundred numbers are stored in consecutive locations starting with location HUND. Find the location of the number with the largest absolute value and store the location of this number at LARGST. Assume that there are no equal numbers.

25. Examine the numbers stored in locations DATA to DATA+49. Determine how many of these numbers are greater than zero and store the count of these numbers in the decrement field of the location ANS. One of the locations contains all zeros. Find this location and place the address of this location in the address field of the location ANS.

26. Given ten numbers: A1, A2, A3, . . . A10, and two other numbers B1 and B2 where B1 > 0 and B2 >

B1. Write a program to compute how many Ai's satisfy the following conditions (A+0 > -0).

- a.  $0 \leq A_i < B_1$
- b.  $B_2 \leq A_i < B_2$
- c.  $A_i \geq B_2$

27. Sort one hundred numbers algebraically in ascending sequence. Stop when no interchange occurs (natural sequence), or when all numbers are sequenced. Numbers are stored in locations NUM through NUM + 99.

28. Write a program to compute:

$$\sum_{i=1}^3 (X_i - Y_i)^2$$

where  $X_i$  and  $Y_i$  are integers. No overflow is to be expected.

### Complement Magnitude - COM ,T

Although not actually an index transmission instruction, the complement magnitude instruction is often used with indexing instructions.

All 1 bits are replaced with 0 bits and all 0 bits are replaced with 1 bits in the  $C(AC)_{Q,P,1-35}$ . The sign position of the AC is unchanged. Since the Y portion of the COM is a part of the operation code, address modification may change the operation.

As an example of COM use, the program shown in Figure 75 shows a table look-up. Given a group of

Location	Operation	Address, Tag, Decrement/Count	Comments	Identification
1 2 4 7 8				72 73 80
ORG	ORG	100	Begin program at location 100.	
	AXT	0,1	Zeros to XR 1.	
	CLA	ARGUE	Place argument into accumulator.	
LOOK	CAS	T,1	Compare first T number with argument.	
	TXI	LOOK,1,-1	AC > T	
	TRA	EQUAL	AC = T	
	TXI	LOOK,1,-1	AC < T	
EQUAL	PXA	0,1	Address of equal T in complement form.	
	COM		Complement this address.	
	ADD	=1	1 bit in position 35 to obtain 2's complement.	
	ADD	LOOK	Instruction with address of first T number.	
	STA	CATCH	Address of T number which agrees	
	HPR		with the argument.	
ARGUE	BSS	1		
T	BSS	100		
CATCH	BSS	1		
	END			

Figure 75. COM Instruction Program Example

numbers in storage locations T1 to T99, find the Ti that agrees with the argument (number being searched for) and store its location in storage location CATCH. Figure 75 shows both the instructions and comments.

### Transfer and Set Index — TSX Y,T

The 2's complement of the location of the *TSX* is placed in the specified *XR*. The computer takes its next instruction from location *Y*.

This instruction may be used to set up a return address to the main program when it is necessary to transfer to a subroutine and, after finishing with the subroutine, to come back to the main program. For example, assume that arithmetic operations are tested for error conditions and, when these conditions are found, a transfer to a fixup routine is to be executed. The last instruction of the fixup routine could be a *TRA* 00001 instruction tagged for the same index register as used by the *TSX* instruction. If the *TSX* is located at core location 1000 and program return to location 1001 is required, the program could be as shown in Figure 76.

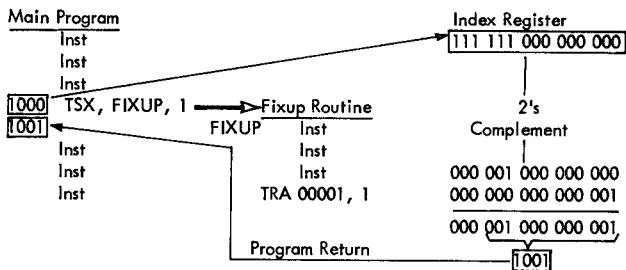


Figure 76. Possible Use of the *TSX* Instruction

### Indirect Addressing

Indirect addressing extends the concept of address modification for a large group of instructions. This extension is carried out in a simple way: just as index registers are "addressed" with a tag, indirect addressing is specified or addressed by a flag (1 bit in both positions 12 and 13 of the instruction). With a flag, the instruction is executed as follows:

1. An effective address is computed in the normal way, by subtracting the contents of the specified index register (if one is specified) from the address part of the instruction. This is called an indirect effective address.

2. The computer then examines the location specified by this indirect effective address and uses the tag and address parts of this word to compute a direct effective address.

The instruction is then executed as if its address part had contained this direct effective address with no tag or flag. The following examples illustrate this process.

Assume that the address part of location 00054<sub>8</sub> in core storage contains 00273<sub>8</sub>. If the instruction *ADD* 00054<sub>8</sub> is executed, the contents of location 00054<sub>8</sub> are added to the contents of the accumulator register. However, if this same instruction has flag bits, the contents of location 00273<sub>8</sub> instead of 00054<sub>8</sub> would be added to the accumulator.

Now, assume further that index registers 1 and 2 contain 4 and 3, respectively, and that core storage location 00050<sub>8</sub> contains a 2 in its tag field and 00167<sub>8</sub> in its address field. If the instruction *ADD* 00054<sub>8</sub> with an index tag of 1 and flag bits is executed, then the indirect effective address equals 00050<sub>8</sub> (address field of the *ADD* instruction minus the contents of index register 1). The direct effective address is 00164<sub>8</sub> (address part of location 00050<sub>8</sub> minus the contents of index register 2), and the contents of this location are added into the accumulator (Figure 77). Remember that flagging always requires positions 12 and 13 of the instruction to contain 1 bits. In text and in program examples, an asterisk represents these 1 bits and indicates that indirect addressing is called for.

### Logic Operations

Logic operations provide means for working on a 36-bit unsigned word or an individual character within a word. All logic operations interpret the sign position of the storage location addressed by the instruction as a numeric bit corresponding to position *P* of the accumulator. The sign position of the accumulator is either ignored or cleared. The instructions to clear, add, and store logical words are:

### Clear and Add Logical Word — CAL Y,T

The *C(Y)* replace the *C(AC)<sub>P,1-35</sub>*. The sign of *Y* appears in *AC<sub>P</sub>* and accumulator positions *S* and *Q* are set to zero. The *C(Y)* are unchanged.

### Add and Carry Logical Word — ACL Y,T

The *C(Y)* are added to the *C(AC)<sub>P,1-35</sub>* and the resultant sum replaces the *C(AC)<sub>P,1-35</sub>*. The sign of *Y* is added to *AC<sub>P</sub>* and a carry from *AC<sub>P</sub>* is added to *AC<sub>35</sub>*. Positions *S* and *Q* of the *AC* are not affected and the *C(Y)* are unchanged.

### Logical Left Shift — LGL Y,T

The *C(AC)<sub>Q,P,1-35</sub>* and *C(MQ)<sub>S,1-35</sub>* are treated as one register and are shifted left the number of places specified by *Y*. The sign of the *AC* is unchanged. Bits enter the *MQ<sub>S</sub>* from *MQ<sub>1</sub>* and go from *MQ<sub>S</sub>* to *AC<sub>35</sub>*. If a



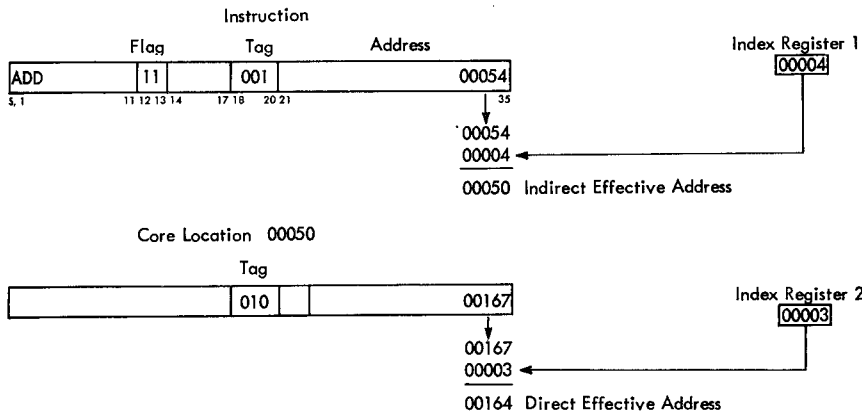


Figure 77. Computing Indirect and Direct Effective Addresses

1 bit is shifted through AC position P, the AC overflow indicator is turned on. Bits are shifted from P to Q and bits shifted from Q are lost. Vacated positions are filled with zeros (Figure 78).

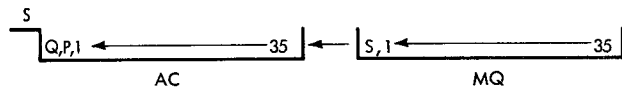


Figure 78. LGL Schematic

#### Logical Right Shift – LGR Y,T

The  $C(AC)_{Q,P,1-35}$  and  $C(MQ)_{S,1-35}$  are treated as one register and shifted right the number of places specified by Y. The AC sign is unchanged. Bits enter  $MQ_S$  from  $AC_{35}$ , and from  $MQ_S$  they are placed in  $MQ_1$ . Bits shifted past  $MQ_{35}$  are lost and vacated positions are filled with zeros (Figure 79).

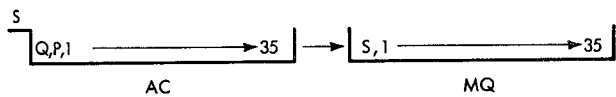


Figure 79. LGR Schematic

#### Logical Compare Accumulator with Storage – LAS Y,T

The  $C(AC)_{Q,P,1-35}$  are treated as an unsigned 37-bit number and are compared with the  $C(Y)_{S,1-35}$ , which are treated as a 36-bit unsigned number. If the  $C(AC)$  are greater than the  $C(Y)$ , the computer takes the next sequential instruction. If the  $C(AC)$  are equal to the  $C(Y)$ , the computer skips the next instruction and proceeds from there. If the  $C(AC)$  are less than the  $C(Y)$ , the computer skips the next two instructions and proceeds from there.

#### Store Logical Word – SLW Y,T

The  $C(AC)_{P,1-35}$  replace the  $C(Y)$ . The P position of the AC is sent to  $Y_S$  and the  $C(AC)$  remain unchanged.

#### Parity Checking Instructions

Two instructions, CAP and SLP, check the parity bit checking circuits to allow operations on an invalid word in the parity trap routine without requesting another parity trap and to enable a program to force a parity trap, thus inhibiting parity checking during special programming situations. Traps are explained in the Trapping section. The format and description of these instructions are:

#### Clear and Add Logical Word with Parity – CAP Y,T

The  $C(Y)_{C,S,1-35}$  replace the  $C(AC)_{S,P,1-35}$ . The parity bit (C) of location Y appears in  $AC_S$  and the sign position of Y appears in  $AC_P$ . Position Q of the AC is set to zero. The  $C(Y)$  are not parity checked and cannot cause a parity trap request. The  $C(Y)$  are unchanged.

#### Store Logical Word with Parity – SLP Y,T

The  $C(AC)_{S,P,1-35}$  replace the  $C(Y)_{C,S,1-35}$ . Position S of location Y is replaced by  $AC_P$ . Unlike all other store operations, parity is not generated during the store; instead, the parity bit of location Y is replaced by  $AC_S$ . The  $C(AC)$  are unchanged. Parity is not checked during the store operation. If an even number of 1 bits are stored in the  $C(Y)$ , any reference to location Y other than a full word store operation or a CAP instruction will result in a parity trap request.

#### Logical Check Sums

One of the principal methods of keeping a check on a block of information in storage is to attach to this block a sum value of all the words in the block. This sum is called a check sum. When computing the sum through

use of logic instructions, the check sum is called a logical check sum. It is normally not equal to the algebraic sum of the block since no overflow occurs with logic instructions.

An example to compute the logical check sum for a block of 300 words in core storage is shown in Figure 80. Normally a symbolic location is assigned to the block of words. For example, the symbol `FIRST` could be used to designate the location of the first word of the block. The symbol `CKSUM` could be used to specify the location where the computed check sum is to be stored.

Another example of check sum computation is shown in Figure 81. Assume five blocks of nine words each. The first block starts at symbolic location `BLOCK + 1`, the second at `BLOCK + 11`, the third at `BLOCK + 21`, and so on. The problem is to find the logical check sum of each block and place it in the first location preceding that block. If the program is started at symbolic location `START`, the instruction sequence could be as shown in Figure 81.

## Packing and Unpacking

There are many cases where the information to be handled by the computer is made up of individual items, each of which is less than the 36-bit computer word. For example, it may be necessary to work with numbers no larger than three decimal digits. To conserve storage space, three such numbers could be stored in the same word (Figure 82), where positions `S`, `14`, and `25` are the sign positions of the numbers `N1`, `N2`, and `N3`, respectively.

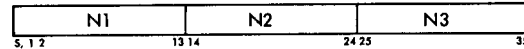


Figure 82. Diagram of Packed Word

Handling information in this way is called packing. In addition to conserving storage space, packing also increases the entry and exit speed of information by reducing, for instance, the amount of magnetic tape to be read or written.

Location	Operation	Address, Tag, Decrement/Count	Comments	Identification
1 2	6 7 8			72 73 80
	<code>AXT</code>	<code>299,1</code>	Load 299 into XR 1.	
	<code>CAL</code>	<code>FIRST</code>	Clear accumulator and add 1st word.	
	<code>ACL</code>	<code>FIRST+300,1</code>	Two instruction loop to compute check	
	<code>TIX</code>	<code>*-1,1,1</code>	sum and test for end condition.	
	<code>SLW</code>	<code>CKSUM</code>	Store computed check sum.	
	<code>HPR</code>		End of routine.	
<code>FIRST</code>	<code>BSS</code>	<code>300</code>	Reserve storage	
<code>CKSUM</code>	<code>BSS</code>	<code>1</code>	locations.	

Figure 80. Check Sum Sample Program

Location	Operation	Address, Tag, Decrement/Count	Comments	Identification
1 2	6 7 8			72 73 80
<code>START</code>	<code>AXT</code>	<code>49,2</code>	Put 49 into XR 2.	
	<code>AXT</code>	<code>9,1</code>	Put 9 into XR 1.	
	<code>PXD</code>	<code>,0</code>	Clear accumulator to zeros.	
	<code>ACL</code>	<code>BLOCK+50,2</code>	Add the block.	
	<code>TNX</code>	<code>*+4,2,1</code>	Test all blocks for end condition.	
	<code>TIX</code>	<code>*-2,1,1</code>	Reduce word count of the block.	
	<code>SLW</code>	<code>BLOCK+40,2</code>	Store the check sum for that block.	
	<code>TIX</code>	<code>START+1,2,1</code>	Test for end of block.	
	<code>SLW</code>	<code>BLOCK+40</code>	Store check sum (last one).	
	<code>HPR</code>		Stop.	
<code>BLOCK</code>	<code>BSS</code>	<code>50</code>	Reserve storage locations.	
	<code>END</code>		End of symbolic instructions.	

Figure 81. Check Sum Sample Program

The AND and OR concept is used, together with a process called masking, to accomplish the packing and unpacking of parts of words. When two numbers are combined with an AND operation, they are matched bit-for-bit. If the same position in each word contains a 1 bit, the result is a 1 bit. If in one word the position is a 0 bit and in the other word it is a 1 bit, the result is a 0 bit. If the same position in both words is a 0 bit, the result is a 0 bit. For example:

```

101101011011  logically added to
101001001101  gives the resulting AND sum of
101001001001

```

An OR function (sometimes called inclusive OR) also matches two numbers bit-for-bit. The difference, however, when compared with an AND, is: (1) if the same position in either word contains a 1 bit, the result is a 1 bit; (2) if the same position in both words is a 1 bit, the result is again a 1 bit; (3) only if the same position in both words is a 0 bit, is the resulting position a 0 bit. For example:

```

011010110101  combined with
001100100100  by the OR operation gives the resulting OR of
01110110101

```

To summarize, mask contents when using the AND operation are 0 bits to unpack and 1 bits to leave data "as are." Mask contents for the OR operation use 1 bits to pack and 0 bits to leave data as are.

### AND to Accumulator — ANA Y,T

Each bit of the  $C(Y)_{S,1-35}$  is matched with the corresponding bit of the  $C(AC)_{P,1-35}$ . The sign position of Y is matched with the  $AC_P$ . When the corresponding bits of both Y and the AC are 1 bits, a 1 bit replaces the contents of that position in the AC. When the corresponding bit of either location Y or AC, or both, is a 0 bit, a 0 bit replaces the contents of that position of the AC. The S and Q positions of the AC are set to zero and the  $C(Y)$  are unchanged. Figure 83 is the flow chart for the ANA instruction.

### OR to Accumulator — ORA Y,T

Each bit of the  $C(Y)_{S,1-35}$  is matched with the corresponding bit of the  $C(AC)_{P,1-35}$ . The sign of Y is matched with  $AC_P$ . When the corresponding bit of either location Y or of the AC, or both, is a 1 bit, a 1 bit replaces the contents of that position in the AC. When the corresponding bits of both location Y and the AC are 0 bits, a 0 bit replaces the contents of that position of the AC. The  $C(Y)$  and the S and Q positions of the AC are unchanged. Figure 84 is the flow chart for the ORA instruction.

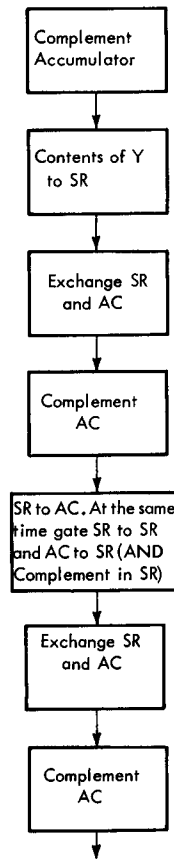


Figure 83. ANA Flow Chart

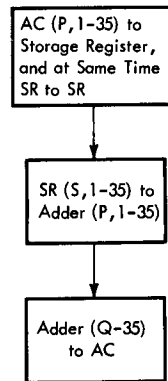


Figure 84. ORA Flow Chart

### ANA Example

As an example in the use of the ANA instruction, assume that a word in core storage has the format shown in Figure 82 and the number N2 is to be operated on. Before arithmetic operations can be performed with this item, it must be separated from the other items in that word location. This separation is called unpacking

or extracting. Since items N1 and N3 are not to be destroyed, the unpacking will be done in the accumulator, leaving the other items intact in core storage. The symbolic program shown in Figure 85 will accomplish this. The mask used in the program contains 1 bits in positions 14-24 (17774<sub>8</sub>) and 0 bits elsewhere. The result of using this mask with the ANA instruction places the N2 number in AC<sub>14-24</sub>. By varying the format of the mask, any of the three numbers could have been unpacked from the packed word.

After performing the desired arithmetic operations on the number N2, a new number, N4, is the result. This number is the same size as N2 and is to be packed (inserted) in location PAKWD, replacing N2, while N1 and N3 are to remain unchanged. The program shown in Figure 86 will accomplish this.

### Adding BCD Coded Numbers

Both the ANA and ORA instructions may be used to perform addition of numbers coded in BCD format. Figure 87 shows the instruction string that accomplishes the addition and the actual bit patterns within the computer.

The bit pattern for the CHS instruction is shown in complement form because changing the sign of the accumulator and then adding CA results in a subtract operation.

### Problem

29. Two packed words, X and Y, contain several small numbers that are distributed within each word as follows:

X1 (S, 1-10)	Y1 (S, 1-10)
X2 (11-17)	Y2 (11-17)
X3 (18-28)	Y3 (18-28)
X4 (29-35)	Y4 (29-35)

The signs of these numbers are in positions S, 11, 18, and 29 of locations X and Y. Write a program to satisfy the following conditions:

- If  $X1 \geq Y1$ , put a 1 bit in location TEST. If  $X1 < Y1$  put a binary 2 in location TEST.
- Same conditions for X2, X3, X4, Y2, Y3, and Y4, using TEST+1, TEST+2, and TEST+3.

Instructions	Bit Patterns
<del>CLA</del> A	001010 000101 000010 000100 000111 001000 (052478)
<del>ACL</del> B	001010 000011 000011 000100 000110 000111 (033467)
ADD SIXES	110110 110110 110110 110110 110110 110110 085945 Ans
<del>SLW</del> STO CA	001010 111110 111011 111111 000100 000101
ANA SIXTY	110000 110000 110000 110000 110000 110000
<del>SLW</del> STO CB	000000 110000 110000 110000 000000 000000
ARS 3	000000 000110 000110 000110 000000 000000
ORA CB	000000 110110 110110 110110 000000 000000
<del>SLW</del> CHS	111111 001001 001001 001001 111111 111111
<del>ACL</del> ADD CA	001010 111110 111011 111111 000100 000101
	001010 001000 000101 001001 000100 000101 (Carry)
	0 8 5 9 4 5

Figure 87. BCD Add Operation Using ANA and ORA Instructions

Location	Operation	Address, Tag, Decrement/Count	Comments	Identification
1 2	6 7 8			72 73 80
	CAL	PAKWD	Place packed word into AC positions P, 1-35.	
	ANA	MASK	N2 is left in AC as a result of ANA operation.	
	ALS	14	Shift N2 until the sign occupies position P.	
	SLW	LOCN2	Store N2 in location LOCN2.	
	HPR		Stop	
MASK	OCT	000017774000	Mask configuration to unpack N2 only.	

Figure 85. Unpacking Program Instructions

Location	Operation	Address, Tag, Decrement/Count	Comments	Identification
1 2	6 7 8			72 73 80
	ARS	14	Shift N4 (N2 after arithmetic operations) to AC 14-24.	
	SLW	LOCN4	Place N4 in temporary storage.	
	CAL	PAKWD	Bring packed word into accumulator.	
	ANA	MASK	Erase N2.	
	ORA	LOCN4	Place N4 in old N2 positions.	
	SLW	PAKWD	Store new packed word.	
	HPR		Stop.	
MASK	OCT	777760003777	Mask configuration to erase N2.	

Figure 86. Packing Program

### Character Handling Operations

Three character handling instructions are used to expedite six-bit character operations. In each of these instructions, positions 15-17 of the instruction itself specify which character of the word located at the effective address (Y) is to be used in the operation. Valid bit patterns for the position field are octal numbers from zero to five and specify the following characters within the word:

OCTAL POSITION FIELD BITS 15-17	WHERE THE CHARACTER TO BE USED IS LOCATED WITHIN THE WORD
0	Positions: S, 1-5
1	6-11
2	12-17
3	18-23
4	24-29
5	30-35
6	See MSM and MIT instructions
7	See MSP and PLT instructions

#### Compare Character with Storage — CCS Y,T,V

The character specified by V and located in Y is compared with the C(AC)<sub>30-35</sub>. If the AC character is greater than the character in Y, the computer takes the next sequential instruction. If the AC character is equal to the Y character, the computer skips the next instruction and proceeds from there. If the AC character is less than the Y character, the computer skips the next two instructions and proceeds from there. The C(AC)<sub>S,Q,P,1-29</sub> are ignored and the C(AC) and C(Y) are unchanged.

#### Place Character from Storage — PCS Y,T,V

The character specified by V and located in Y replaces the C(AC)<sub>30-35</sub>. The C(AC)<sub>S,Q,P,1-29</sub>, and the C(Y) are unchanged.

#### Store Accumulator Character — SAC Y,T,V

The C(AC)<sub>30-35</sub> is placed in location Y in the character position specified by V. The remaining bits of Y and the C(AC) are unchanged.

As an example in the use of the PCS and SAC instructions, assume that a word consisting of six alphanumeric characters (9-code) is located in storage location FRWRD. The instruction sequence shown in Figure 88 would create a word of the same six characters in reverse order and store these characters in storage location BKWRD.

#### Problem

30. A parts purchase record in core storage consists of three words:

Word 1. Part number (six alphanumeric characters, 9-code)

Word 2. Quantity (binary integer)

Word 3. Price per unit in cents (binary integer)

There are five types of parts, distinguished by the last character (positions 30-35) of the part number. Write a program to compute the total money invested in each part type by summing the individual calculations of price times quantity for each part type. Parts types are A, C, F, J, and R. There are no invalid part types. There are 9,000 parts purchase records located in storage locations INVPR through INVPR 8999. Place the part type totals in locations ATOTL, CTOTL, FTOTL, JTOTL, and RTOTL.

#### Data Transmission

The ability to move blocks of information from one series of storage locations to another set of storage locations is provided by one instruction, TMT.

#### Transmit — TMT Y,T

This instruction uses the C(AC)<sub>3-17</sub> as a FROM address and the C(AC)<sub>21-35</sub> as a TO address. The contents of the FROM location in core storage replace the contents of the TO location and the C(FROM) remain unchanged. The

Location		Operation	Address, Tag, Decrement/Count	Comments	Identification		
1	2	4	7	8	72	73	80
		PCS	FRWRD,,0	Character in FRWRD positions S, 1-5 are			
		SAC	BKWRD,,5	placed in BKWRD positions 30-35.			
		PCS	FRWRD,,1	FRWRD position 6-11 are placed in			
		SAC	BKWRD,,4	BKWRD positions 24-29.			
		PCS	FRWRD,,2	FRWRD positions 12-17 are placed in			
		SAC	BKWRD,,3	BKWRD positions 18-23.			
		PCS	FRWRD,,3	FRWRD positions 18-23 are placed in			
		SAC	BKWRD,,2	BKWRD positions 12-17.			
		PCS	FRWRD,,4	FRWRD positions 24-29 are placed in			
		SAC	BKWRD,,1	BKWRD positions 6-11.			
		PCS	FRWRD,,5	FRWRD positions 30-35 are placed in			
		SAC	BKWRD,,0	BKWRD positions S, 1-5.			

Figure 88. PCS and SAC Instruction Use

C(FROM+1) then replace the C(TO+1) and the C(FROM+1) remain unchanged. This process continues until the total words specified by the Y part of the TMT instruction have been transmitted.

Positions 28-35 of the TMT specify the number of words to be moved and provide a maximum transfer of  $377_8$  or  $255_{10}$  words. Any number larger than  $377_8$  is interpreted as modulo  $400_8$ . Modulo  $400_8$  means that, given a transmit count, the actual number of words moved will be the remainder after dividing the count by  $400_8$ . With indexing, the count number is modified (positions 28-35 only) by the specified index register. A step-by-step description shows that:

1. Positions 28-35 of the TMT are modified by the specified XR (if any) and then are placed in the shift counter.
2. If the contents of the shift counter are zero, the instruction ends.
3. With a non-zero value in the shift counter, the C(AC)<sub>3-17</sub> are used to address core storage.
4. The C(Y) specified by the C(AC)<sub>3-17</sub> are placed in the storage register.
5. The C(AC)<sub>21-35</sub> are used to address core storage.
6. The C(SR) are placed in the location specified by the C(AC)<sub>21-35</sub>.
7. The C(AC)<sub>3-17</sub> and C(AC)<sub>21-35</sub> are both increased by one.
8. The shift counter is reduced by one.
9. Execution returns to step 2.

At the completion of the TMT, the C(AC)<sub>3-17</sub> contain the address of the last word read, plus one. The

C(AC)<sub>21-35</sub> contain the address of the last word stored, plus one. Another TMT instruction can be given if it is desired to transmit more than  $377_8$  words. As an example in possible use of the TMT, assume that 512 words stored in locations AREA1 are to be relocated to storage locations AREA2. The program could be as shown in Figure 89.

### Problems

31. Given a block of  $100_{10}$  words in core storage, in storage locations  $1750_8$  through  $2114_8$ , move the first  $25_{10}$  words to locations 100, 101, etc.; move the second block of  $25_{10}$  words to locations 200, 201, etc.; the third block of 25 words to locations 300, 301, etc.; and the fourth block to locations 400, 401, etc.

32. Given three blocks of data located in storage and containing:

Block 1 =  $45_{10}$  words

Block 2 =  $30_{10}$  words

Block 3 =  $15_{10}$  words

Make one block of  $90_{10}$  words in consecutive storage locations.

### Floating-Point Operations

When the range of numbers anticipated during a calculation is either large or unpredictable, it becomes difficult to work with fixed-point arithmetic instructions. An alternative set of floating-point instructions is available for such calculations. These instructions maintain the binary point automatically.

Location	Operation	Address, Tag, Decrement/Count	Comments	Identification
	<i>A X T</i>	<i>512, 1</i>	Put 512 into XRI.	
	<i>C L A</i>	<i>STADD</i>	Get Starting address.	
	<i>T M T</i>	<i>1</i>	Transmit one word.	
	<i>T I X</i>	<i>*-1, 1, 1</i>	Test for end; if not end, reduce and repeat.	
	<i>H P R</i>		Stop.	
<i>S T A D D</i>	<i>P Z E</i>	<i>AREA1, AREA2</i>	Start addresses for From and To.	
	<i>E N D</i>		End of symbolic instructions.	
		Another variation might be:		
	<i>C L A</i>	<i>STADD</i>	Get starting address.	
	<i>T M T</i>	<i>255</i>	Transmit 255 words.	
	<i>T M T</i>	<i>255</i>	Transmit 255 words.	
	<i>T M T</i>	<i>2</i>	Transmit 2 words.	
	<i>H P R</i>		Stop.	

Figure 89. Sample Transmit Programs

A floating-point decimal number X may be expressed as a signed proper fraction (N) multiplied by some integral power (n) of 10. The number is normal if the power of 10 (n) is chosen so that the decimal point is positioned to the left of the most significant digit of N. Examples:

$$\begin{array}{rcl} x & N & 10^n \\ -.010 & = & -.10 \times 10^{-1} \\ .140 & = & .14 \times 10^0 \\ 4.600 & = & .46 \times 10^1 \\ 88.000 & = & .88 \times 10^2 \end{array}$$

Likewise, a floating-point binary number (X) may be represented as a signed proper fraction (B) times some integral power (b) of 2. In the normalized case, the binary point is positioned to the left of the most significant digit of B. Examples:

$$\begin{array}{rcl} x(\text{BINARY}) & B(\text{BINARY}) & 2^b(\text{DECIMAL}) \\ -.001 & = & -.100 \times 2^{-2} \\ .100 & = & .100 \times 2^0 \\ 1.100 & = & .110 \times 2^1 \\ 110.000 & = & .110 \times 2^3 \end{array}$$

The algebraic addition of two floating-point numbers in the computer is analogous to the ordinary algebraic addition of two signed numbers with decimal points. An example is the algebraic addition of the numbers 100 and  $-0.1009$ :

$$\begin{array}{r} 100.0000 \\ -000.1009 \\ \hline 99.8991 \end{array}$$

Note that the second number must be shifted to the right to line up the decimal points, and that the first number must be supplied with additional zeros. The same addition performed with numbers expressed in floating-point decimal form is:

$$\begin{array}{r} .1000 \times 10^3 \\ -.1009 \times 10^0 \\ \hline \end{array}$$

Again, before the addition, the lower number is shifted to the right with a compensating change in the exponents, and corresponding zeros are added to the number on the upper line:

$$\begin{array}{r} .1000000 \times 10^3 \\ -.0001009 \times 10^3 \\ \hline .0998991 \times 10^3 = .998991 \times 10^2 \end{array}$$

Note also that the digits of the answer must be moved to the left to be in normalized form (no zero in the position to the right of the point) and that the final fraction contains more digits than either of the two numbers involved in the addition.

In the computer, the two numbers are expressed as binary fractions, each having an eight-bit binary characteristic to represent the exponent 2. The "lining up" is done by shifting from the AC into the MQ. The result

of an addition or multiplication is normalized by shifting the fractions in the AC and MQ left while making compensating changes in the characteristic of the sum or product.

In the computer, a floating-point number is stored in a word location as shown in Figure 90.

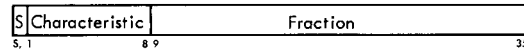


Figure 90. Floating-Point Word Format

The fraction is contained in bit positions 9 through 35. The sign of the fraction is contained in the S position of the word, and position 1 of the characteristic may be considered the sign of the characteristic. For example, an exponent of  $-32_{10}$  would be represented by a characteristic of  $200_8$  minus  $40_8$  or  $140_8$ . An exponent of  $100_{10}$  would be represented by a characteristic of  $200_8$  plus  $144_8$  or  $344_8$ . Since  $128_{10}$  is equal to  $200_8$ , the characteristic of a non-negative exponent always has a 1 bit in position 1 of the floating-point word, while the characteristic of a negative exponent always has a 0 bit in position 1. A normal zero has no bits in either the characteristic or the fraction, and is the smallest possible zero available in this notation.

### Conversion

A procedure for converting numbers to floating-point notation can be illustrated by the problem: Convert the decimal fraction .149 to floating-point notation:

1. Convert to binary form:

$$.149_{10} = .1142_8 = .001\ 001\ 100\ 010_2$$

2. Enter the binary number into the fraction part of the word with a zero ( $200_8$ ) characteristic:

$$10\ 000\ 000.\ 001\ 001\ 100\ 010\ \text{or}\ (200.\ 1142)_8$$

3. Normalize:

$$01\ 111\ 110.\ 100\ 110\ 001\ \text{or}\ (176.\ 461)_8 = \text{answer}$$

Now, convert the decimal integer 149 to floating-point notation:

1. Convert to binary form

$$149_{10} = 225_8 = 010\ 010\ 101$$

2. Strike out leading zeros

$$10\ 010\ 101$$

3. Enter this binary number into the fraction part of the word with a zero characteristic

$$10\ 000\ 000.\ 100\ 010\ 101\ \text{or}\ (200.\ 452)_8$$

4. Add the octal number of binary digits in step 2 to the zero characteristic of the computer word

$$10\ 001\ 000.\ 100\ 101\ 010\ \text{or}\ (210.\ 452)_8 = \text{answer}$$

Examples of equal exponential (binary) and floating-point numbers are:

EXPONENTIAL BINARY FORM (BINARY EXPONENTS)	FLOATING-POINT FORM				
	S	I	8	9	35
$1 \times 2^{11}$	0	10	000 011	100 000 000	.0
$1 \times 2^{11}$	0	10	000 100	100 000 000	.0
$-101 \times 2^1 = (-0.101 \times 2^{100})$	1	10	000 100	101 000 000	.0
$.11 \times 2^9$	0	10	000 000	110 000 000	.0
$1 \times 2^{-11} = (.1 \times 2^{-10})$	0	01	111 110	100 000 000	.0
$-.101 \times 2^{-1011}$	1	01	110 101	101 000 000	.0

The fraction does not always have a significant bit to the right of the binary point but, when it does, the floating-point number is said to be in "normal" form. The exception to this rule is a "normal" zero. A normal zero is a floating-point number whose characteristic and fraction are both zero. If a floating-point number does not meet either of these qualifications, it is called "unnormal." The single-precision floating-point instructions are divided into the two general categories: normal and unnormal. The difference in machine operation between the two is that normal operation always attempts to produce a normal answer and unnormal does not.

### Examples

**Multiplication: Example 1.** Add characteristics and multiply fractions.

S	I	8	9	35		
0	10	000 011	.11	...	0	Multiplicand.
0	10	000 001	.101	...	0	Multiplier.
<hr/>						
		100 000 100	.01111	...		Add characteristics and multiply fractions.
		10 000 000				Subtract extra 200 <sub>s</sub> factors from characteristic.
		10 000 100	.01111	...	0	Normalize by shifting fractions left one place and decreasing characteristic by one; this does not alter the value of product.
		10 000 011	.1111	...	0	

**Multiplication: Example 2.**

S	I	8	9	35		
0	01	111 011	101	...	0	Multiplicand.
1	10	000 110	1100	...	0	Multiplier.
<hr/>						
1	100	000 001	.01111			Add characteristics and multiply fractions.
		10 000 000				Subtract extra 200 <sub>s</sub> factor in characteristic.
1	10	000 001	.01111			Normalize by shifting fraction left one place and reducing the characteristic by one.
1	10	000 000	.1111			

The sign of the product is negative because the signs of the two original factors were different.

Normalization of one place is automatic on the FMP instruction, whether or not the multiplier or multiplicand is in normal form. If both factors are normal, floating-point multiply will produce a normal product. Normalization is not performed by the computer on unnormalized floating-point multiply (UFM) operations,

regardless of whether the factors are normal. UFM cannot produce a normal product.

**Division:** Divide fractions and subtract characteristics.

S	I	8	9	35		
0	10	001 010	.1000	...	0	Divide fractions and subtract characteristics
0	10	000 101	.1000	...	0	Add 200 <sub>s</sub> (FP factor)
<hr/>						
		10 000 000				Shift fraction right one place (point must be to left of most significant figure)
0	10	000 101	1.000	...	0	
0	10	000 110	.1000	...	0	
<hr/>						
Proof: (Decimal) $.5 \times 2^6 = 1 \times 2^5 = 32$						
$\quad \quad \quad .5 \times 2^5$						
(Binary) $.1 \times 2^5 = 2^5 = 32$						

Preceding the division, the dividend is in the AC and the divisor in the SR. The MQ is automatically cleared before division takes place. After division the quotient appears in the MQ and any remainder in the AC.

If both the dividend and divisor are in normal form, the quotient will be in normal form. When the dividend or divisor is not in normal form, the quotient will be normal only if the fraction of the dividend is greater than half but less than twice the fraction of the divisor.

**Addition and Subtraction:** As in fixed point, subtraction is accomplished by inverting the sign of the SR.

**Floating Point Binary: Example 1.**

S	I	8	9	35		
0	10000011	.101	...	0	Signs different; subtraction is implied.	
1	10000011	.100	...	0		
<hr/>						
		10000011	.0010	...	0	Characteristics are equal; therefore subtract.
		10000001	.100	...	0	Fractions assign same characteristic.
<hr/>						
Proof (Decimal) $2^3 \times .625$						
$\quad \quad \quad -2^3 \times .500$						
<hr/>						
$2^3 \times .125 = 8 \times .125 = 1.0$						
$1.0 = 10000001 .10 \dots 0$						

**Example 2.**

S	I	8	9	35		
0	01	111 000	.001010	...	0	
0	10	000 101	.111000	...	0	Signs are alike; addition is implied.
<hr/>						
LOWEST CHARACTERISTIC						
01	111 000	.001010				The characteristics must be made equal by shifting the fraction of the lowest number to the right and increasing the characteristic by the number of shifts.
01	111 001	.000101				
01	111 010	.0000101				
01	111 011	.00000101				
01	111 100	.000000101				
01	111 101	.0000000101				
01	111 110	.00000000101				
01	111 111	.000000000101				
10	000 000	.0000000000101				
10	000 001	.00000000000101				
10	000 010	.000000000000101				
10	000 011	.0000000000000101				The characteristics are now equal; therefore add fractions and affix common characteristic.
10	000 100	.00000000000000101				
10	000 101	.000000000000000101				
10	000 101	.111000000000000000				
10	000 101	.111000000000000101				



In addition or subtraction, the characteristics must be made equal before the fractions can be combined. The number with the smallest characteristic is automatically placed in the accumulator. Then it is shifted right a number of places equal to the difference in the AC and SR characteristics. Bits shifted past AC<sub>35</sub> go to MQ<sub>9</sub>, and bits leaving MQ<sub>35</sub> are lost. Normalization of the total occurs on FAD and FSB.

### Summary

Floating-point arithmetic is used to reduce programming complexity and increase the range of numbers available for calculation. The only disadvantage is the loss of two and one half decimal places of accuracy (lost in accommodating the characteristic).

The only major difference in exponential and floating-point arithmetic is the treatment of the exponent. Negative exponents are implied by floating-point characteristics of less than 200<sub>8</sub>.

#### Multiplication:

Add characteristics and reduce by 200<sub>8</sub>

Multiply fractions and normalize

#### Division:

Subtract characteristics and increase by 200<sub>8</sub>

Divide fractions and normalize

#### Addition and Subtraction:

Equalize characteristics by shifting the fraction having the smallest characteristic right, at the same time increasing the characteristic proportionately. Combine fractions (add or subtract) and normalize.

#### Sign Control:

Multiplication and Division: Factors' signs alike; answer plus.

Factors' signs unlike; answer minus.

Addition and Subtraction: Answer always has sign of largest factor.

The possibility of floating-point overflow or underflow during execution of a floating-point instruction is indicated by an (\*) asterisk in the following descriptions. All conditions of underflow and overflow are discussed following the last floating-point instruction and are also included under "Trapping."

## Single-Precision Floating-Point Instructions

### Floating Add — FAD Y,T

The floating-point numbers located in Y and the AC are added together. The most significant portion of the result appears as a normal floating-point number in the AC. The least significant portion of the result appears in the MQ as a floating-point number with a

characteristic 33 (octal) less than the AC characteristic. The signs of the AC and MQ are set to the sign of the larger factor. The sum in the AC and MQ is always normalized whether the original factors were normal or not. If C(AC)<sub>1-35</sub> contain zeros, the FAD may be used to normalize an unnormal floating-point number.

1. The MQ register is cleared to zeros.

2. The C(Y) are placed in the SR.

3. If the characteristic in the SR is less than the characteristic in the AC, the C(SR) and C(AC)<sub>8,1-35</sub> are interchanged, as the number with the smaller characteristic must appear in the AC before addition can take place.

4. The MQ is given the same sign as the AC.

5. If the difference in the characteristics is greater than 63, the C(AC) are cleared. If the difference in the characteristics is a number *N* less than or equal to 63, the C(AC)<sub>9-35</sub> are shifted right *N* places. Bits shifted out of position 35 of the AC enter position 9 of the MQ. Bits shifted out of position 35 of the MQ are lost.

6. The characteristic in the SR replaces the C(AC)<sub>1-8</sub>.

7. The C(SR)<sub>9-35</sub> are added to the C(AC)<sub>9-35</sub> and this sum replaces the C(AC)<sub>9-35</sub>. If the signs of the AC and SR are unlike, the C(SR)<sub>9-35</sub> are added to the 1's complement of the C(AC)<sub>9-35</sub>. Since the C(AC)<sub>9-35</sub> represent a pure fraction, the magnitude of their 1's complement is equal to  $(1 - 2^{-27}) - C(AC)_{9-35}$ .

8. Regardless of the sign or relative magnitudes of the SR and AC, the result appears in double-precision form with signs alike in both the AC and MQ. If the signs of the AC and SR are the same and the magnitude of the sums of the fractions is greater than or equal to one, there is a carry from position 9 into position 8 of the AC. Thus, the characteristic of the AC is increased by one. In this event, the fractions of the AC and MQ are shifted right one position and a 1 is inserted into position 9 of the AC. If the signs of the AC and SR are different, there are two cases, both depending on the difference between the SR and AC fractions.

CASE 1. If the magnitude of the SR fraction is greater than the fraction in the AC, the AC and MQ signs are both changed to the sign of the SR. If the fraction of the MQ is zero, the difference between the fractions of the SR and AC is placed in the AC. If the fraction of the MQ is not zero, the difference between the fractions of the SR and AC, minus one, is placed in the AC; the 2's complement of the MQ fraction replaces the fraction in the MQ.

CASE 2. If the magnitude of the SR fraction is less than the fraction in the AC, the difference of the two fractions replaces the fraction of the AC. The sign of the AC and the entire MQ remain unchanged.

9a. If the resulting fractions in both the AC and MQ are zero, the AC is cleared, yielding a normal zero. If the fractions are in normalized form before the FAD is given, this result can only occur if the signs are different and the  $C(Y)_{1-35}$  are equal to the  $C(AC)_{1-35}$ . The signs of the AC and MQ will be equal to the sign of the number originally in the AC. If the resulting fraction in the AC is zero and the two numbers were not in normalized form before addition, the signs of the AC and MQ are equal to the sign of the original number having the smaller characteristic.

9b. If the resulting fractions in the AC and MQ are not zero, the fractions of the AC and MQ are shifted left until a 1 appears in position 9 of the AC. Bits enter position 35 of the AC from position 9 of the MQ. The characteristic in the AC is reduced by one for each position shifted. No shifting is necessary if the fraction of the AC is in normal form at the beginning of this step.

10. The MQ is given a characteristic which is 27 less than the characteristic in the AC, unless the AC contains a normal zero, in which case zeros are left in positions 1-8 of the MQ.

If the P and/or Q positions of the AC are not zero before the execution of the FAD, the result will usually be incorrect. Non-zero bits in P and/or Q which are initially interpreted as part of the AC characteristic make it larger than the characteristic in the SR so that the interchange in step 3 will always take place. During the interchange a 1 will be placed in position S of the SR if there is a 1 in either S or P positions of the AC, so that the sign of the number may be changed. Any bit in Q is lost during the interchange and both P and Q are cleared when the  $C(SR)$  replace the  $C(AC)$ . The difference between the two characteristics is computed after the interchange occurs, so that in step 5, N will not be equal to the difference between the original characteristics. In step 6 the characteristic in

the SR, with its Q and P bits missing, replaces the characteristic in the AC. Consider as a sample problem the addition of:

$$\begin{aligned} 2^2 \times .1001 &= (SR) + 10000101.1001 \\ 2^5 \times .1001 &= (AC) + 10000101.1001 \end{aligned}$$

First, the exponents must be equalized and then the addition may proceed. The characteristics are checked and found unequal, with the largest in the AC. The numbers in the AC and SR are then exchanged, giving:

$$\begin{aligned} SR &+ 10000101.1001 \\ AC &+ 10000101.1001 \end{aligned}$$

The MQ content is zeros at this time. The  $C(AC)_{9-35}$  are then shifted right the number of places needed to equalize the exponents. (Remember that the binary point is located between positions 8 and 9 of all registers.) The registers then appear as:

$$\begin{aligned} SR &+ 10000101.1001 \\ AC &+ 10000101.0001 \\ MQ &+ 00000000.0010 \end{aligned}$$

The fractions (positions 9-35) may now be added.

$$\begin{aligned} SR &+ 10000101.1001 \\ AC &+ 10000101.1010 \\ MQ &+ 00000000.0010 \end{aligned}$$

AC position 9 is checked for a 1 and no normalizing occurs. The MQ characteristic is now set. It is equal to the AC characteristic minus the number of places in the AC fraction (27 in the computer, 4 in this example):

$$\begin{aligned} SR &+ 10000101.1001 \\ AC &+ 10000101.1010 \\ MQ &+ 10000001.0010 \end{aligned}$$

Decoding the results into the original format, we find:

$$\begin{array}{r} 2^5 \times .1001 \quad MQ = 2^1 \times .0010 = 2^5 \times .00000010 \\ 2^5 \times .0001001 \quad AC = \quad \quad \quad 2^5 \times .1010 \\ \hline 2^5 \times .1010001 \quad \text{resultant sum} = 2^5 \times .10100010 \end{array}$$

The FAD instruction may be used to convert a fixed integer to a floating integer of less than  $2^{27}$  through use of a program as shown in Figure 91.

Location		Operation	Address, Tag, Decrement/Count	Comments	Identification
1	2	6	7	8	72 73 80
		C.L.A.	FIXED	Put fixed integer into accumulator.	
		O.R.A.	CHAR		
		F.A.D.	CHAR		
		S.T.O.	FLOAT	Store converted integer.	
		H.P.R.			
C.H.A.R.		O.C.T.	233000000000	(Octal number)	

Figure 91. FAD Conversion Sample Program

**Floating Divide or Proceed – FDP Y,T**

The C(AC) are divided by the C(Y). The quotient appears in the MQ and the remainder appears in the AC. If the magnitude of the AC fraction is greater than or equal to twice that of the C(Y)<sub>9..35</sub>, or if the magnitude of the C(Y)<sub>9..35</sub> is zero, division does not occur and the computer takes the next instruction in sequence. The quotient is in normal form if both the dividend and divisor are in normal form. The sign of the MQ is the algebraic sign of the quotient. If the AC fraction is zero, the C(AC)<sub>Q,P,1..35</sub> are cleared and the AC sign is set plus. The C(Y) are unchanged.

**Floating Multiply – FMP Y,T**

The C(Y) are multiplied by the C(MQ). The most significant part of the product appears in the AC and the least significant part appears in the MQ. The product of two normalized numbers is in normalized form. If either of the numbers is not normalized, the product may or may not be in normalized form. The C(Y) are unchanged.

**Floating Subtract – FSB Y,T**

This instruction algebraically subtracts the number located in Y from the number in the AC and normalizes the result. The C(Y) are unchanged.

**Unnormalized Floating Add – UFA Y,T**

This instruction algebraically adds the two numbers contained in the AC and located by Y. The sum is not normalized and the C(Y) are unchanged.

The UFA instruction may be used to convert floating-point numbers to fixed-point numbers if the magnitude of the floating point number is less than 2<sup>27</sup>. The instruction sequence could be as shown in Figure 92.

**Unnormalized Floating Multiply – UFM Y,T**

This instruction multiplies the number at Y by the number in the MQ. The result is not normalized and the C(Y) are unchanged.

**Unnormalized Floating Subtract – UFS Y,T**

This instruction algebraically subtracts the number at Y from the number in the AC. The result is not normalized and the C(Y) are unchanged.

**Trapping**

Automatic trapping of the program is used with the 7040/7044 systems to signal unusual conditions to the program without special test instructions. With trapping, system status is constantly monitored and, when special conditions are detected, normal processing is interrupted and the program is transferred (trapped) to a trap routine.

To identify the causes of trapping and to allow for a return to normal processing, the instruction counter contents are automatically stored at a fixed location in storage, usually with some trap identification data, when a trap is initiated. The program is then automatically transferred to another fixed core storage

Location		Operation	Address, Tag, Decrement/Count	Comments	Identification		
1	2	6	7	8	72	73	80
		ORG		Start of program.			
		CLA	FLOAT	Place FP number in accumulator.			
		UFA	CHAR				
		ALS	10				
		ARS	10				
		STO	INT	Store integer.			
		LGL	B				
		LRS	0	Put AC sign in MQ.			
		STQ	FRAC	Store fraction.			
		HPR					
CHAR		OCT	233000000000,0,0,				
FRAC							
INT							
FLOAT							
		END					

Figure 92. UFA Conversion Sample Program

location for its next instruction. Many types of trapping are used and each type is assigned a priority with regard to the other types. Only the floating-point trap is presented here. All other traps and the special trapping instructions are described under "Trapping."

### Floating-Point Trap

During the execution of floating-point instructions, the resultant characteristic in the AC and MQ may exceed eight bit positions (result is too large for storage). The capacity is exceeded if the exponent goes beyond  $177_8$  or below  $-200_8$ . Beyond  $177_8$  is termed overflow; below  $-200_8$  is termed underflow. Overflow and underflow may occur in either the AC or the MQ registers. The computer, on sensing an underflow or overflow, will put the address, plus one, of the instruction that caused the condition, into the address part of location 00000. An indication of the actual cause (a spill) is stored in the decrement part of location 00000. The decrement bit positions used and their meaning when they contain a 1 bit is:

BIT	MEANING
S	Double-precision instruction on system with single-precision only.
12	Double-precision address error
14	Single-precision divide instruction
15	Overflow in AC or MQ or both
16	AC overflow or underflow
17	MQ overflow or underflow

After the storing of the trap information, the computer automatically executes the instruction located in location 00010.

If a trap to location 00010 occurs and the contents of location 00000 are  $0000120000375_8$ , the instruction that caused the trap was a floating-point divide instruction (a 1 bit in position 14). The cause of the trap itself was an underflow in the accumulator (0 bits in positions 15 or 17 and a 1 bit in position 16) and the next instruction following the floating-point divide in normal sequence is in location  $00375_8$  (positions 21-35 of location 00000).

### Double-Precision Floating-Point Instructions

Four double-precision floating-point instructions are available for applications requiring higher accuracy than possible with single-precision instructions. These instructions increase floating-point precision from 8 to 16 decimal digits by working with two full 36-bit words at a time. All double-precision numbers in core storage must be located so that the high-order word is in an even address core location followed by the low-order word in the next higher odd address location. If the

effective address of a double-precision instruction is odd, the instruction is trapped (explained under "Trapping").

All rules described for single-precision floating-point instructions also apply to the double-precision floating-point instructions. For overflow and underflow, the exponent of the major or high-order word of the result (in the AC) may not exceed  $+177_8$  and the exponent of the minor or low-order word (in the MQ) may not be lower than  $-200_8$ . Double-precision instructions executed on a system with single-precision option only, cause a trap operation.

Figure 93 shows the format of double-precision floating-point words, both in the processing unit registers and in core storage locations.

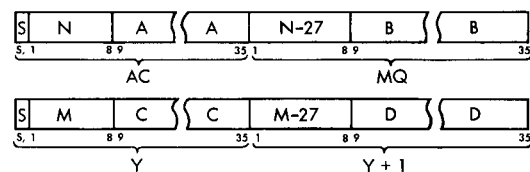


Figure 93. Double-Precision Word Format

### Double-Precision Floating Add — DFAD Y,T

This instruction adds the number located in Y and Y+1 to the number in the AC and MQ. The result is a normalized double-precision number with the major answer in the AC and the minor in the MQ. The signs of the AC and MQ are the algebraic sign. The contents of Y and Y+1 are unchanged.

### Double-Precision Floating Subtract — DFSB Y,T

This instruction is equivalent to DFAD with the sign in Y inverted.

### Double-Precision Floating Multiply — DFMP Y,T

This instruction causes the number in Y and Y+1 to be multiplied by the number in the AC and MQ. The result is a normalized number in the AC and MQ, with an associated algebraic sign. The  $c(Y)$  and  $c(Y+1)$  are unchanged.

### Double-Precision Floating Divide or Proceed — DFDP Y,T

This instruction causes the number in the AC and MQ to be divided by the number in Y and Y+1. The result is a double quotient in the AC and MQ, with an associated algebraic sign. If the magnitude of the AC fraction is greater than or equal to twice that of the  $c(Y)_{9-35}$ ,

or if the  $CY^{9-35}$  are zero, the divide check indicator is turned on and the computer takes the next sequential instruction.

**Problem**

33. Write a trap routine to be used with a floating-point arithmetic program. The program is only concerned with the quotient in division and with the most significant portion of the answer in other floating-point arithmetic operations. The trap routine should:

- a. Place a correctly signed zero in location 01000 if there is an underflow in the most significant portion of the answer.
- b. The routine should halt if there is a double-precision address error or if there is an overflow in the most significant portion of the answer.
- c. Both the AC and MQ should be left unchanged if the indicated error occurred in the least significant portion of the answer.

## IBM 7040/7044 Input/Output Control System

With the high internal processing speeds of modern computers, most applications involving large amounts of data handling are input/output limited; that is, the time consumed by input/output devices in transmitting data to and from core storage is very large when compared with the time necessary to process that data. Therefore, much effort has been expended to make input/output as fast as possible. In the 7040 and 7044 systems, such features as multiple data channels, overlapping input and output with processing, and data channel traps are direct results of the need for rapid input/output.

These advanced concepts require careful programming for efficient use. Even the simplest program requires extensive coding if all input/output facilities of the computer are to be used. Moreover, the resulting routines resemble one another, so the programmer is often duplicating work already accomplished elsewhere.

To avoid this duplication of effort, a standard set of input/output routines has been written. These routines contain features that no single programmer would have time to write on his own and are thus more flexible than an input/output routine written for a single application. The aggregate of these routines is the Input/Output Control System (IOCS).

The most important feature of IOCS, aside from simplification of input/output operations, is the provision for symbolic reference to input/output units. Instead of referring to a specific input/output device on a specific channel, the programmer refers to a system unit function. The device assigned to perform this function can be changed by reassembly of the IBSYS Basic Monitor, under which IOCS operates, or by the programmer. Thus, the uses made of input/output devices may be varied to increase efficiency and speed job-to-job transitions. IOCS also provides standardized error recovery routines for each device that can be attached to the 7040 and 7044 systems.

### Basic Concepts

The data that the programmer manipulates is usually in the form of a file. A file is a collection of related information arranged in logical records. A logical record may consist of a single number, may be all of the information pertaining to a given business transaction, or may be a record of the value of several parameters at a given point in an experiment.

The main problem in the usual data processing application is that of performing certain calculations with the data contained in the files for that application. Most files, however, are much too large to be held entirely in core storage; it is necessary to read part of the file, process it, and write out the results. Since a logical record is the smallest amount of data that a program can read and still have enough information to begin calculating, the program is mainly concerned with the handling of logical records. Sometimes logical records are too small for efficient recording on tape or disk storage. It then becomes necessary to block records, by putting two or more logical records together, before recording them. IOCS provides routines that simplify the problems of blocking and deblocking logical records.

### Labels

A label is a single block, at the beginning or end of a data file, which describes the file. The label at the beginning of the file is the header label. The trailer label, the last block in the file, comes after the end-of-file mark which indicates the last data record has been read.

IOCS contains routines to verify and create standard 120-character IBM header and trailer labels on input and output files. All header labels begin with a file character code and contain information such as: the name of the file, the date of creation of the file, the file retention period, and the parity and density at which the file was written. Trailer labels begin with other codes to indicate that the file continues on a different input/output device or that the end of the data file has been reached.

Labels are also used for checking, to insure that the right file is read, that it is read in the correct parity, and that files containing data that should be retained are not erased.

### Overlap and Data Channels

Data channels control transmission of information to and from an input/output device selected by the processing unit. When channel A is in operation, all processing is delayed until data transmission is complete. Channels B through E, however, allow the processing unit to continue with calculation while data are being transmitted. All four of these channels can carry on input/output operations simultaneously, overlapping channel operations and those of the processing unit.

### **Channel Traps**

The most efficient use of overlapped channels is obtained when they are kept in continuous operation. Although the main program can determine by a transfer instruction whether an input/output operation has been completed, periodic checking of the channel-in-operation indicator takes time and complicates programming. To avoid this situation, the 7040 and 7044 systems can trap the program as soon as any channel operation ends.

A trap causes the operation of the main program to be suspended, and the contents of the instruction counter to be stored in the address portion of one of the first few words of core storage. Indications about the reason for the trap and the conditions encountered by the channel during the operation are also placed in the same location. The next instruction is then taken from the location immediately following the one in which the information was stored. This instruction transfers control to a supervisory routine, which chooses the select and error recovery routine for the device concerned and causes it to be executed. This routine checks for errors in transmission, institutes error correction procedures if necessary, and starts new activity, if possible. The supervisor then restores the traps and returns control to the main program.

A fundamental part of IOCS is a trap supervisor, which makes it unnecessary for the programmer to write supervisory routines. In fact, the trap supervisor and the trap locations in lower core storage are under storage protection, and any attempt to modify them halts the offending program.

### **Buffers**

Effective overlapping of data transmission with processing requires at least two input/output areas (buffers), for each file. At a given time, one buffer holds the logical record or records being processed, and the other holds the record or records being read or written. Because the buffers reverse roles as soon as the processing and input/output operations are complete, buffering involves switching procedures within the main program. This may become complicated if several files from different devices are being processed against one another.

Also, since each buffer contains a single block, it may contain several logical records. Separating these records for individual processing can also make the program more involved. IOCS contains routines that handle the use of buffers and the blocking and deblocking of logical records within the buffers.

### **IOCS Organization**

The input/output control system is divided into four sections:

*Input/Output Buffering System — IOBS:* Routines necessary for buffering input/output and for blocking and deblocking logical records.

*Input/Output Operations — IOOP:* Routines for starting activity and error correction procedures on the devices attached to the data channels. IOOP is also responsible for the scheduling activity of individual devices.

*Input/Output Labeling System — IOLS:* Routines that read, write, check, and construct labels.

*Input/Output Executor — IOEX:* A trap supervisor, a channel scheduler, a series of conversion and utility routines, and a scheduler of special routines to be executed as soon as an input/output operation is completed.

### **IOCS Level Concept**

Although each section of IOCS performs a specific operation, the sections are not independent. IOBS, IOOP, IOLS, and IOEX combine in various ways to form three distinct levels of IOCS.

The IOBS level involves the existence in core storage of all four sections of IOCS. The lower levels are all used by IOBS and, if IOBS is used on a given file, none of the lower levels may be used on the same file. The program specifies the file characteristics to IOBS and then can view them as continuous strings of logical records that enter or leave core storage on demand. IOBS uses IOLS according to the specifications of the file.

At the IOOP level, the programmer loses the blocking, deblocking, and buffer supervision facilities of IOBS. He communicates directly with IOOP, and also with IOLS. The purpose of IOOP is the reading and writing of blocks.

The IOEX level, although it still permits the programmer to use IOLS, mainly provides him with trap supervision and channel scheduling. The programmer must provide the select and error recovery routines normally provided by IOOP.

### **Relationship of IOCS Levels**

IOBS contains most of the coding necessary for efficient use of all input/output capabilities of the 7040 and 7044 systems. If the IOOP or IOEX levels are used instead of the IOBS level, machine coding must be supplied to make up the deficit between the facilities available at the level being used and those necessary for efficient operation of the program.

At the IOOP level, blocking and deblocking and buffer supervision routines necessary to handle the blocks obtained through IOOP must be supplied. This may be the most efficient course if the files being handled do not require elaborate blocking and deblocking. It is also possible that the data are not arranged in sequential

files, as on disk storage, in which case IOBS is not applicable.

At the IOEX level, routines to take the place of IOOP, in addition to the routines necessary to replace IOBS must be written. The extra routines include a unit scheduler and a select and error recovery routine for each type of device that the program uses. The IOEX level is useful to reduce the amount of space taken up by input/output routines to the minimum possible for the particular application. It may also be used in circumstances for which IOOP is not appropriate.

### **Relationship of IOCS to IBSYS**

IOCS is part of the IBSYS Operating System. The input/output executor and the parts of IOOP required by the systems library loader are kept in core storage at all times, along with the nucleus of the basic monitor. These two programs are storage protected to avoid damage to them from untested programs (if the memory protect option is a part of the system).

The remaining sections of IOCS are loaded by the monitors and left in core storage for object programs. Any part that is not required by an object program may be overlaid except where that part is storage protected. These sections are used by the program through calling sequences to the entry points of the various levels of IOCS.

In using IOCS, the programmer refers to the devices being used by the logical name of the function that they perform, not by their actual machine addresses. When IOCS receives a request for some input/output operation, it obtains the machine address of the attached device that can perform the system unit function specified. This address is used to carry out the instructions received from the main program. IOCS itself is entirely device-independent, and allows for the possibility that any one of a large number of devices may have been attached to a given system unit function. It is the responsibility of the operator to insure that the physical devices attached to the system are compatible with the use to which they are being put.

### **Random and Sequential Processing**

IOCS contains facilities for both sequential and random processing applications. In sequential processing, the files have a fixed sequence and are read in and processed according to that sequence. Card and tape files are naturally sequential, and IOCS provides for disk storage to be used for sequential processing.

In random processing, the data files being processed are not necessarily in sequence with respect to each other. One file is read sequentially, and the sequence of other files that must be processed at the same time is indicated by the records of that file. All files that are in a different order from the file being read sequentially must be held on some random access input/output device such as disk storage. IOOP includes routines to facilitate random processing. These routines are device-oriented, since they require that a random access device be attached to the system unit function to which they are applied. Also, the program must give IOOP the location on the device where the logical record of data can be found.

### **Summary**

The user of IOCS must determine at what level the best use may be had of the 7040 and 7044 systems and of the monitor facilities provided with the system.

The greatest programming flexibility (and consequently the greatest effort required) and the least core storage space sacrifice are available at the lowest IOCS level. The closest approach to logical data handling and the greatest storage space requirement occur at the highest IOCS level.

The intermediate levels of IOCS may be the most satisfactory for programs whose record format is completely known and whose buffering requirements are easily coded, but which are completely independent of the input/output devices available.

IOCS, therefore, provides each user with a program package that can fulfill his individual application requirements. To use IOCS, a 7040/7044 Data Processing System must have the extended performance instruction set option.



## Input/Output Devices and Operations

To use the fast, versatile processing ability of the processing unit, the user must be able to put raw data into the computer system, tell the system what to do, and take the processed data from the system and record it in a form that can be used. This chain of input, processing, and output always begins and ends with some input/output device.

An input/output device is a machine linked directly to the data processing system. Each device operates under control of the processing unit as directed by the stored program. In some cases, a separate unit, placed between the processing unit and the input/output device, serves as a control or synchronizer unit. The synchronizer not only controls the devices attached to it but serves as an assembly-disassembly device for the data passing through it. This is necessary since the internal processing speeds are much faster than the input devices that read data or the output devices that record the results. Figure 94 shows the relationship between processing unit, synchronizer, and input/output devices.

Input devices sense or read data from IBM cards, magnetic tape, paper tape, or may simply supply data to the processing unit in the form of electronic pulses. The data are then placed in the core storage of the system. Output devices record or write the data from storage on IBM cards, magnetic and paper tape, or prepare printed copy. Output may also be in the form of electronic pulses (for transmission over communications networks).

### Reading and Writing

Reading takes place as the input medium physically moves through an input device. The information is sensed or read and is converted to a form that may

be used by the computer system. The information is then sent to core storage.

Writing involves converting data from storage to a form or language compatible with an output medium and recording the data using an output device.

Most input/output devices are automatic; once started, they continue to operate as directed by the stored program. Instructions in the program select the required device, direct it to read or write, and indicate the storage location that data will be put into or taken from. A few input devices are manually operated, and no medium for recording data is involved. Instead, data are entered directly into the computer using a keyboard or switches, which are usually a part of an operator's console.

### Data Buffering

All data processing procedures involve input, processing, and output. Each phase of the procedure takes a specific time. The usefulness of a computer is often directly related to the speed at which it can complete a given procedure. Ideally, the configuration and speed of the various input/output devices should be so arranged that the processing unit is always kept busy with useful work. The efficiency of any computer system can be increased to the degree in which input, output, and internal processing can be overlapped or allowed to occur simultaneously.

Figure 95 shows the basic time relationship between input, processing, and output with no overlap of operations. In this type of data flow, processing is suspended during reading or writing operations.

Figure 96 shows an overlapped time relationship. The figure assumes that there are two buffers, one for input and another for output. With this type of data

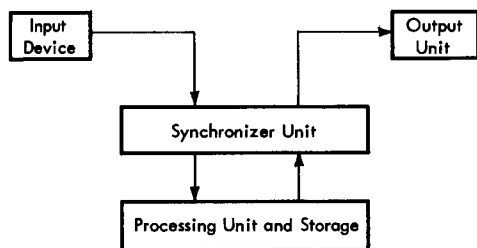


Figure 94. Input/Output Device Relationship

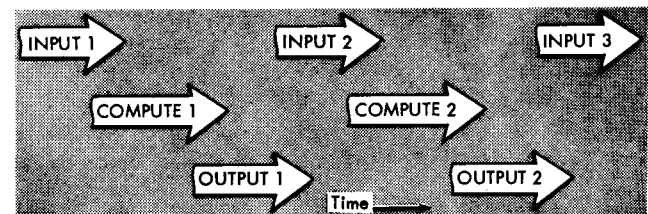


Figure 95. Non-Overlap Time Relationship

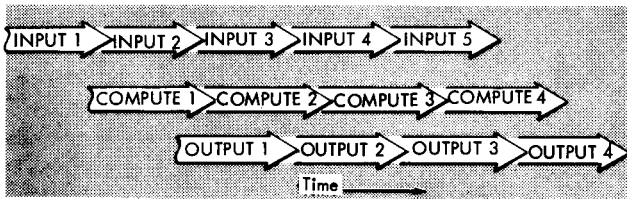


Figure 96. Overlapped Time Relationship

buffering system, data are first collected in an input buffer. When called for by the program, the contents of the input buffer are sent to core storage. The transfer takes only a fraction of the time that would be required to read the data directly from an input device. Also, while data are being assembled in the buffer, processing can occur in the processing unit. Likewise, completed data from storage can be placed in an output buffer at high speed. The output device is then instructed to write out the contents of this buffer. While writing occurs, the processing unit is free to continue with other work.

With the 7040 and 7044 systems, both the nonoverlap and overlapped types of operation are available. For small computer applications, requiring only card and printer equipment, the basic or nonoverlap system could be used. If higher input/output volume is required, magnetic tape can be attached to the system, still using nonoverlapped operation. If still higher job completion time (throughput) is needed, the overlapped data buffering method may be used. Magnetic tape units, Tele-Processing® equipment, and other types of input/output devices may be attached to the overlapped system.

Actually, input/output devices of the 7040 and 7044 systems are linked to the processing unit with a data channel. Registers within the data channel control the quantity and the destination of all data transmitted between storage and the input/output devices. The basic system of the 7040 and 7044 computers includes one input/output channel, data channel A, whose operation is not overlapped with processing unit operation. Overlapped input/output and processing unit operation is available by using the IBM 7904 Data Channel. Figure 97 shows a 7040 or 7044 system, using only data channel A and some of the available input/output units. By using the 7904 Data Channel, a multiple channel system is possible, which expands to include many different input/output devices in various combinations (Figure 98).

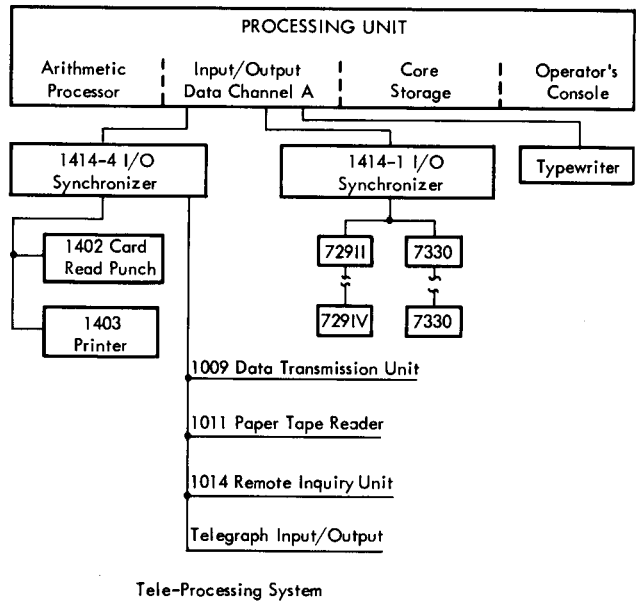
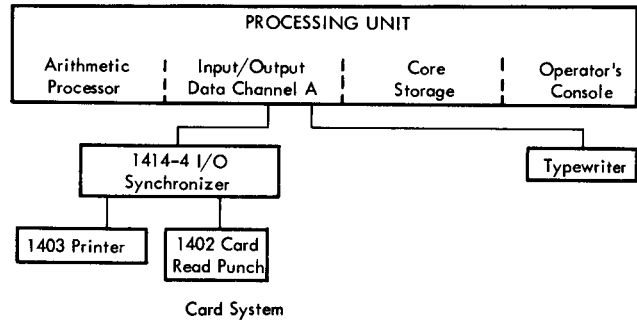


Figure 97. Channel A Systems

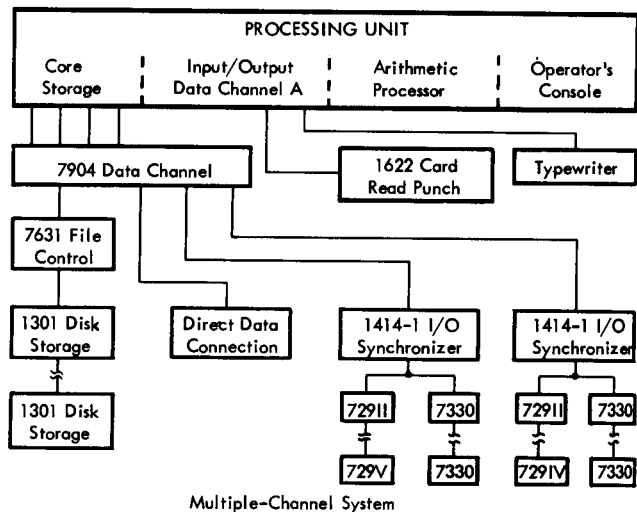
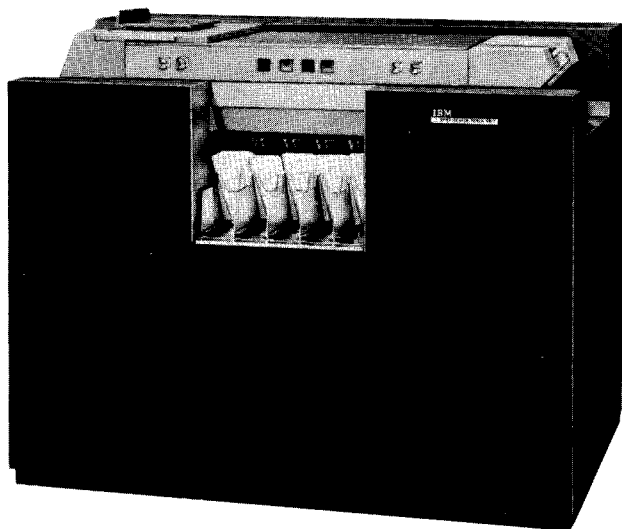


Figure 98. Multiple Channel System

## Devices and Control Units used on 7040 and 7044 Systems

Many optional input/output device configurations are available on the 7040 and 7044 systems. Both the control units and actual input/output devices are presented in list form; the input/output devices are then described in Figures 99 through 108. Some channel A devices do not require a control unit but are connected directly to an adapter of channel A.

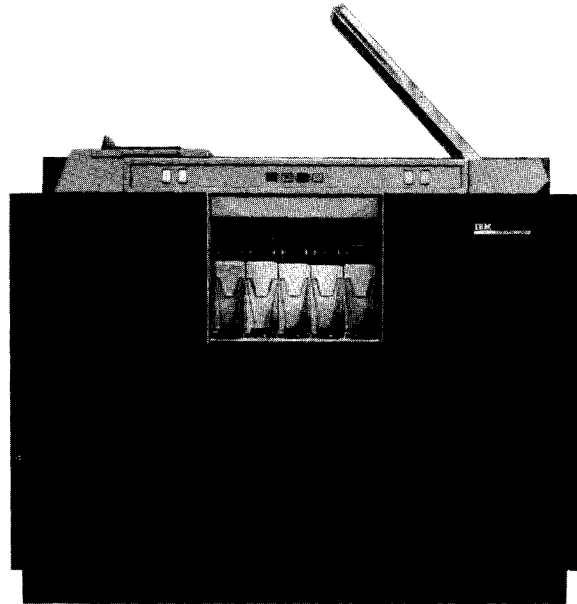
CONTROL UNIT	DEVICE
IBM 1414-1 I/O Synchronizer	IBM 729 II Magnetic Tape Unit
	IBM 729 IV Magnetic Tape Unit
	IBM 729 V Magnetic Tape Unit (with 800 CPI Feature)
	IBM 7330 Magnetic Tape Unit (with Tape Intermix Feature)
IBM 1414-2 I/O Synchronizer	IBM 7330 Magnetic Tape Unit
	IBM 1402-2 Card Read Punch
IBM 1414-3 I/O Synchronizer	IBM 1403-1 or -2 Printer
	IBM 1402-2 Card Read Punch
IBM 1414-4 I/O Synchronizer	IBM 1403-1 or -2 Printer
	IBM 1009 Data Transmission Unit
IBM 1414-5 I/O Synchronizer	IBM 1011 Paper Tape Reader
	IBM 1014 Remote Inquiry Unit
	Telegraph-type Units
	Column Binary Feature (for 1402)
	IBM 1009 Data Transmission Unit
	IBM 1011 Paper Tape Reader
IBM 1414-7 I/O Synchronizer	IBM 1014 Remote Inquiry Unit
	Telegraph-type Units
IBM 1414-7 I/O Synchronizer	IBM 729 II Magnetic Tape Unit
	IBM 729 IV Magnetic Tape Unit
	IBM 729 V Magnetic Tape Unit
	IBM 729 VI Magnetic Tape Unit
	IBM 7330 Magnetic Tape Unit (with Tape Intermix Feature)



Reads 250 cards per minute  
Punches 125 cards per minute  
Attached directly to Data Channel A  
One 1622 per 7040/7044 system

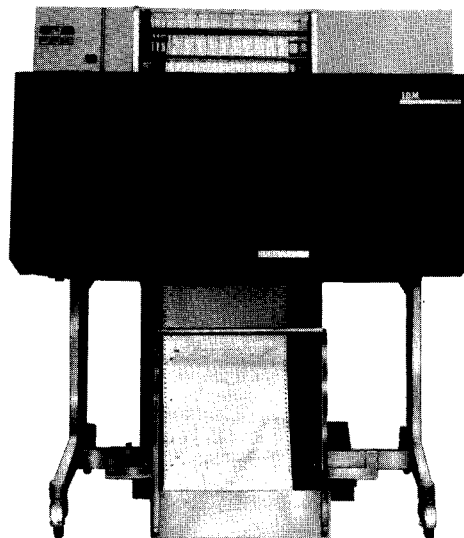
Figure 99. IBM 1622 Card Read Punch

A maximum of ten tape units (all models) may be attached to any 1414-1, -2, or -7 Input/Output Synchronizer. The IBM 1622 Card Read Punch and the IBM 1401 Data Processing System do not require an input/output synchronizer. A console typewriter is a



Reads 800 cards per minute  
Punches 250 cards per minute  
May have Column Binary Feature  
Attached to 1414-3 or -4  
One 1402 per 1414

Figure 100. IBM 1402 Card Read Punch



Maximum of 132 printing positions per line  
Prints 600 lines per minute  
Tape controlled carriage  
Attached to 1414-3 or -4  
One 1403 per 1414

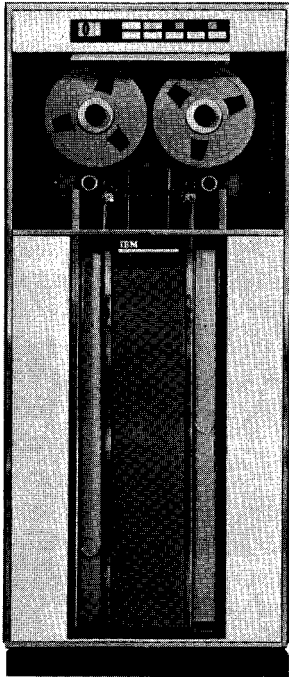
Figure 101. IBM 1403 Printer

standard feature on both the 7040 and 7044, and it does not require a synchronizer.

Input/output control units (or devices) that may be attached to channel A include one each of the following:

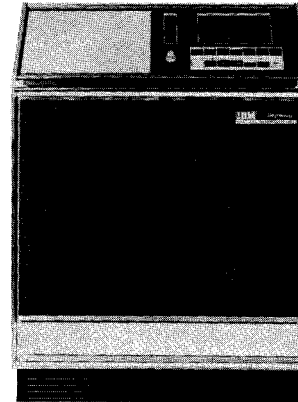
1. One console typewriter.
2. One IBM 1414-1, -2, or -7 Input/Output Synchronizer and its attached magnetic tape units.
3. One IBM 1414-3, -4, or -5, Input/Output Synchronizer and its attached units, or one 1622 Card Read Punch.
4. One IBM 1401 Data Processing System, Models B through F, and its attached input/output units.

Four IBM 7904 Data Channels may be attached to the 7040 or 7044 system. Each 7904 has one input/



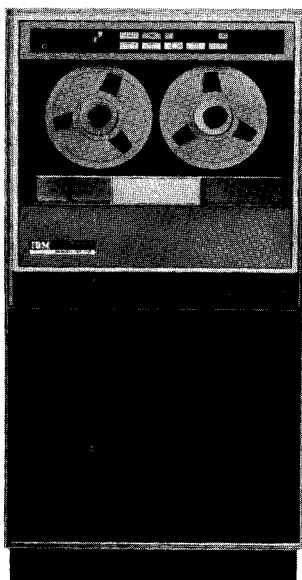
Character Rates (characters per second):  
 729 II and 15,000 cps, or  
 729 V 41,667 cps, or  
 60,000 cps  
 729 IV and 22,500 cps, or  
 729 VI 62,500 cps, or  
 90,000 cps  
 729 II and V move tape at 75 inches per second  
 729 IV and VI move tape at 112.5 inches per second  
 All 729 models attach to 1414-1 or -7  
 Maximum of ten units per 1414

Figure 102. IBM 729 II Magnetic Tape Unit



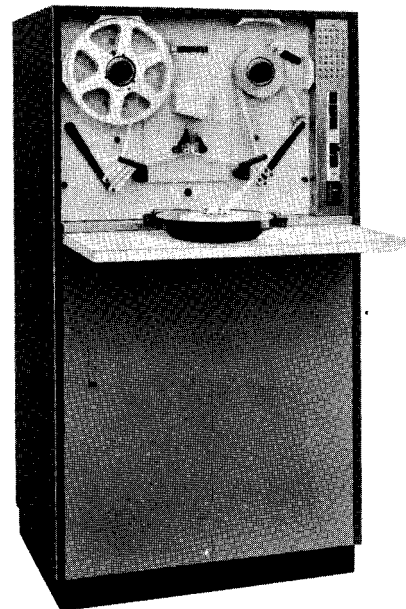
Data rates of 75, 150, 250, or 300 characters per second  
 Connects two computers; each computer must have its own 1009  
 Attached to the 1414-4 or -5

Figure 104. IBM 1009 Data Transmission Unit



Character Rates (characters per second):  
 7,200 cps, or  
 20,016 cps  
 Tape moves at 36 inches per second  
 Attached to 1414-2 or the 1414-1 or -7 with Tape Intermix Feature  
 Maximum of ten tape units per 1414, including 729 tape units on 1414-1 or -7

Figure 103. IBM 7330 Magnetic Tape Unit



Character rate of 500 paper tape characters per second  
 Either 5-track or 8-track paper tape  
 Tape may be chad or chadless in strips, reels, or rolls  
 Attached to 1414-4 or -5  
 One 1011 per 1414

Figure 105. IBM 1011 Paper Tape Reader

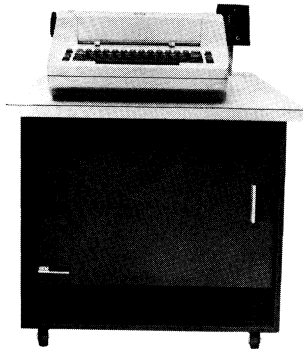
output control adapter. This control adapter allows attachment of any one of the input/output devices designed to the input/output control adapter interface specifications. Interface is defined as the actual lines and functions designed into such devices, including the actual signal, data, and control lines (and their precise cable connections), together with the necessary internal functions. This means that future input/output

devices designed to these same specifications may be attached to the 7904 with a minimum of effort. In addition to the one input/output control adapter, each 7904 Data Channel may have one of the following devices:

1. One IBM 1414-1, -2, or -7 Input/Output Synchronizer and its attached tape units.
2. One Direct Data Connection (Explained later). This feature provides for connection of non-IBM input/output devices, such as analog/digital converters, radar, microwave links, etc.

The 7904 input/output control adapter allows attachment of any one of the following devices:

1. One IBM 7631 File Control and its associated IBM 1301 Disk Storage.
2. One IBM 7750 Programmed Transmission Control and its attached input/output devices.

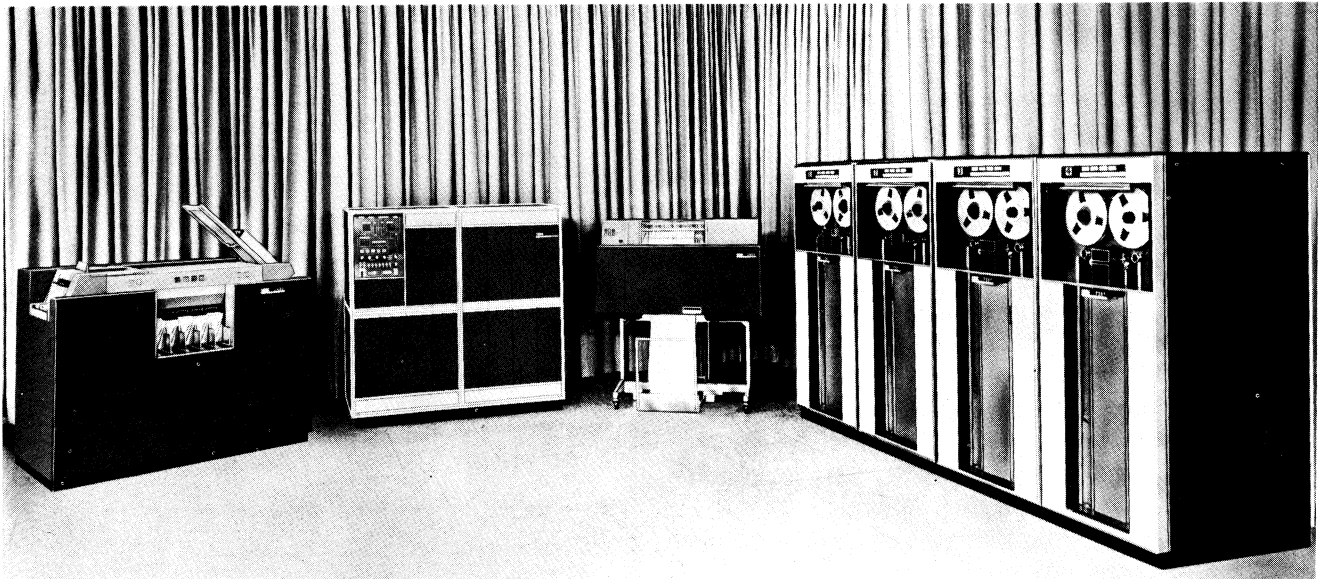


Maximum data rate of 15 characters per second  
 Maximum of 78 characters per message  
 Used to interrogate computer from remote location  
 Maximum of 20 units per 1414  
 Attached to 1414-4 or -5

Figure 106. IBM 1014 Remote Inquiry Unit

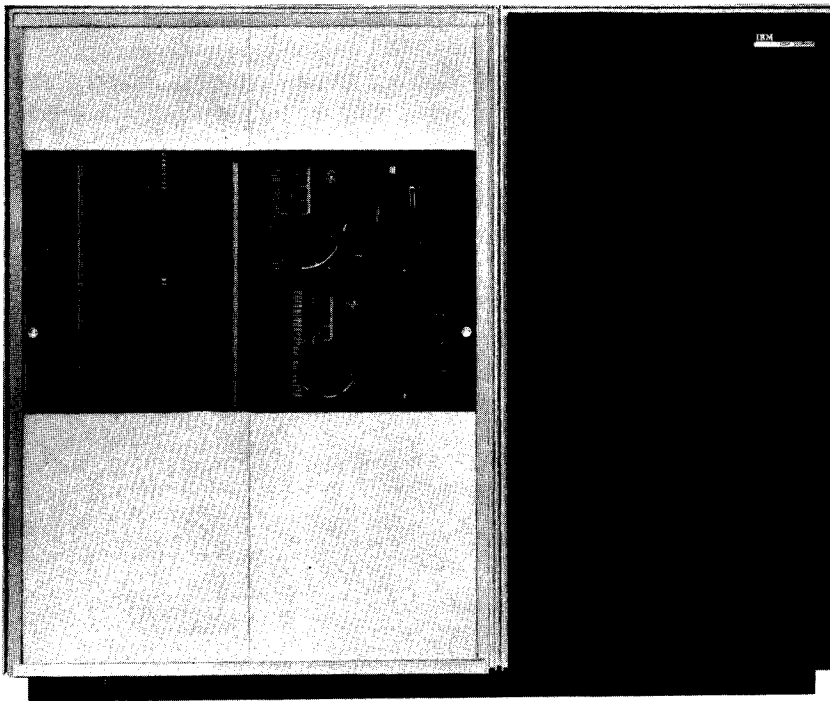
### Data Channels

Data channels of the 7040/7044 systems use a command word technique. With this technique, control of an input/output operation passes logically from the processing unit to the data channel. The command word is the means of transferring control from processing unit to data channel and, therefore, the data channel must have enough registers and counters to exercise this control. The channel is required to



Considered as an input/output device by the 7040/7044 system  
 Except for input/output instructions, each computer's instructions function normally  
 Both systems can operate together on a single problem or independently (with the 1401 off-line) on different problems  
 Each system may interrogate the busy status of the other system  
 Attached directly to Data Channel A.

Figure 107. IBM 1401 Data Processing System



Maximum of five 1301 units per 7040/7044  
 Maximum capacity of 55,800,000 characters of storage  
 per 1301  
 Controlled by one or two 7631 File Controls  
 Attached to the 7904 input/output control adapter

Figure 108. IBM 1301 Disk Storage

perform such functions as word counting, address changing, and (with some devices) the assembly and disassembly of words of data.

**Data Channel A Operation**

Data channel A uses registers and data paths of the processing unit to perform the input/output device control function and therefore, no overlap exists between input/output and computer operations. Figure 109 shows data flow for channel A.

An input/output operation is started by execution of a select instruction, which is decoded by the processing unit. The output of the decoders is sent to the channel to select the input/output adapter and the input/output device specified by the select instruction. If the device is busy with other work or is not ready for operation, the select waits until the device is free.

Upon execution of the select, the input/output device called into use actually starts moving. When selection is successfully completed, the channel ends operation on the select and the processing unit gets the next instruction for execution. This instruction is normally a reset and load channel (RCH) instruction. The RCH specifies a storage location that contains the

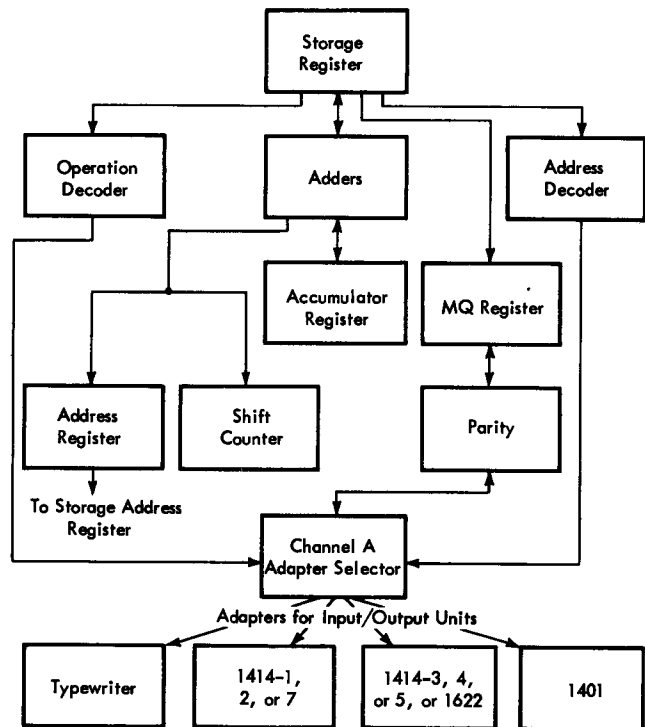


Figure 109. Data Flow for Channel A

data channel command for this input/output operation. One RCH exists for each data channel.

Understanding the relationship between the select and the RCH instructions is very important. The speed of input/output devices is much slower than processing unit execution speeds; however, the program must have executed a RCH instruction by the time the particular device being used is ready to read or write. This is shown in Figure 110, using a read select instruction for a 729 II Magnetic Tape Unit. A maximum of 4 milliseconds may exist in the program between execution of the RDS to a 729 II tape unit and execution of the RCH. The RCH may, however, immediately follow the RDS or be placed anywhere within the time allowed. To further point out the time relationship: the 4 milliseconds needed to get the tape unit moving and actually reading data into the processing unit is enough time for execution of 500 machine cycles on a 7040 system or about the time required to execute 200 average-time instructions.

Operation	Comments
RDS 729 II Tape ..... ..... ..... .....	Execution of the RDS starts physical motion in the selected tape unit.
RCH..... .....	Four (4) milliseconds after the RDS is executed, the RCH must be executed; otherwise, the tape unit is disconnected from the read operation.

Figure 110. Time Relationship between Select and RCH Instructions

Execution of the reset and load channel instruction places the data channel command in the accumulator. The command (Figure 111) specifies the number of words to be moved (word count — positions 3-17) and the first storage address (start address — positions 21-35) to be used for the read or write operation. Read operations are defined as input; write operations are defined as output.

Operation	Word Count	Starting Address
5 1	2 3	17 18 20 21 35

Figure 111. Data Channel Command Format

The word count portion of the command is tested for zero count and, when it is zero, the channel ends operation on the command and disconnects the input/output device. Disconnect means that the input/output device is no longer needed (has finished its operation) and may be released for further use. When the word count is not zero, the starting address part is placed

in the address register (Figure 109), and the shift counter is set to six. At this point, operation differs for read and write operations.

#### READ OPERATION

The input/output device sends seven-bit bytes (six data bits plus a parity bit) to the channel adapter where the parity of the byte is checked. If a parity check is detected, the channel redundancy check indicator is turned on but the input/output operation continues. The six data bits are sent to the multiplier-quotient register (MQ) and placed in positions 30-35. The contents of the MQ are shifted left six positions and the shift counter (set to six initially) is reduced by one. This procedure continues until the shift counter is reduced to zero, which indicates that a full 36-bit word (six bytes) has been transferred to the MQ. The contents of the MQ are then placed in the storage register and stored at the address specified by the starting address.

Command modification is now accomplished by routing the starting address to the address, increasing this address by one, and returning the address to accumulator positions 21-35. The word count is likewise reduced by one and returned to accumulator positions 3-17. If the word count is zero, the channel stops reading into storage and, when an end-of-record signal is received from the input device, ends operation on the channel command and disconnects the input device. If the word count is not zero, the shift counter is again set to six and the channel repeats the procedure.

As input bytes come to the MQ register, the parity bit is removed and accumulated to check word parity.

#### WRITE OPERATION

A word (36 bits) is brought from the location specified by the starting address and is placed in the MQ. Positions S, 1-5 are sent to the parity generating circuits, and the parity bit is added to the six data bits. The complete byte is then sent to the output device, through the channel adapter selector. After transmission of the first six data bits, the MQ is shifted left six positions and the shift counter is reduced by one. This procedure continues until the shift counter is reduced to zero, indicating that 36 data bits have been transferred (as six bytes) to the output device.

Command modification occurs as with a read operation and the word count is tested for zero. If it is zero, the channel ends operation and disconnects the output device. If it is not zero, a new data word is required from core storage; therefore, the modified address is sent to the address register and the shift counter is again set to six. This procedure continues until the word count equals zero.

### IBM 7904 Data Channel Operation

The IBM 7904 Data Channel is used with the 7040 and 7044 Data Processing Systems to provide an overlapped compute and input/output program operation. Up to four 7904 Data Channels may be attached to the computer and are designated as data channels B through E. These data channels incorporate four registers to perform their function. Figure 112 shows data flow within the data channel. The registers used include:

*Channel Data Register:* This 37-position register acts as a buffer between core storage and the assembly register. The data register has inputs from the storage bus, direct data, and assembly register on a full-word basis.

*Word Counter:* This 15-position counter contains the number of words to be transmitted to or from the data channel. The counter is loaded from the storage bus (positions 3-17) before data transmission begins and is decreased by one for each word processed.

*Address Counter:* This 15-position counter contains the starting address in core storage of the information to be stored or transmitted. The counter is loaded from the storage bus (positions 21-35) before data transmission begins and is increased by one for each word processed.

*Assembly Register:* This 36-position register serves as a buffer between the channel data register and input/output equipment. Data are assembled and disassembled in the register for transmission.

Read and write operation, using the 7904 Data Channel, is much the same as with data channel A. The outstanding difference is that central processing unit registers are not used with the 7904 and, therefore,

the CPU and the 7904 Data Channels can operate independently. The operation is simply started by the CPU and taken over by the 7904. The CPU is then signalled when the 7904 has completed the operation or when an error signal is received by the 7904, so that the CPU can use the data from the input device or put the channel to work on other operations.

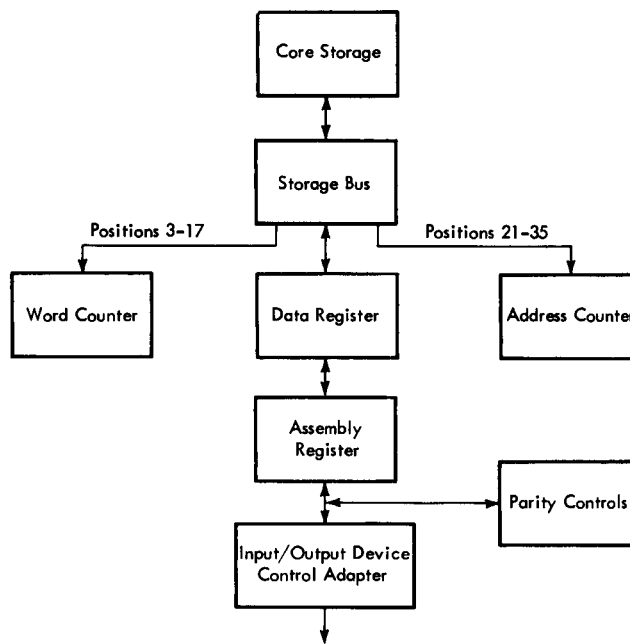


Figure 112. IBM 7904 Data Channel Data Flow



## IBM 1414 Models 1, 2, and 7

These models of the 1414 perform a magnetic tape control function for the 7040 and 7044 systems. Up to ten magnetic tape units may be attached to any 1414 Models 1, 2, or 7:

- IBM 1414-1     IBM 729 II and 729 IV tape units; 729 V (if the 1414 is equipped with the 800 CPI feature) and 7330 if equipped with the Intermix Feature.
- IBM 1414-2     IBM 7330 tape units.
- IBM 1414-7     IBM 729 II, IV, V, VI tape units; 7330 if equipped with the Intermix Feature.

The tape units operate in either binary or BCD modes under program control, as specified in the address part of the select instruction.

## Magnetic Tape

IBM magnetic tape is similar to the tape used in home tape recorders. It is a plastic tape, 1/2 inch wide, and coated on one side with a metallic oxide. Data are recorded as magnetized spots or bits in the metallic oxide. Information recorded on tape is permanent and can be retained for an indefinite time. Previous recordings are destroyed as new information is written. This means that tape can be used repetitively with significant savings in recording costs. Several types of magnetic tape are available to meet varying requirements of strength, durability, reliability, and cost.

For handling and processing, tape is wound on plastic reels containing up to 2,400 feet of tape (lengths as short as 50 feet may be used). The magnetic tape unit, which functions both as an input and output device, moves the magnetic tape and accomplishes the actual reading or writing of information on the tape. Data are recorded in seven parallel channels or tracks along the tape. Seven bit positions across the width of

the tape (one in each channel) provide one column of data. The spacing between columns of bits is automatically established by the magnetic tape unit used in writing.

Records of data on tape may range from one or two characters to several thousand. The size of the record is limited only by the length of tape or the capacity of the storage units that data will be placed in or removed from.

## Seven-Bit Alphameric Code

The seven recording tracks or channels on tape are labeled C, B, A, 8, 4, 2, 1 and correspond to the seven bit positions of the seven-bit alphameric code. A character is represented by the presence or absence of bits in the seven channel positions of one column, across the width of the tape. Figure 113 shows characters in the seven-bit alphameric code as they appear on tape.

To verify tape reading and writing, each character is checked for even parity. In addition to this vertical parity check, a horizontal (longitudinal) parity check is made on each record. At the time a record is written, the bits in each horizontal row are counted. At the end of the record, a check character is recorded. This character has a bit corresponding to each channel row with an odd bit count. Thus, when the record is read, each channel row of the complete record, including the check character, should satisfy the even parity condition. The check character serves this purpose only and is never included as part of the record when data are transferred to the computer system.

Tape written in the seven-bit alphameric code can be used by several data processing systems, providing a means of intercommunication from one system to another. There are instances, however, where special

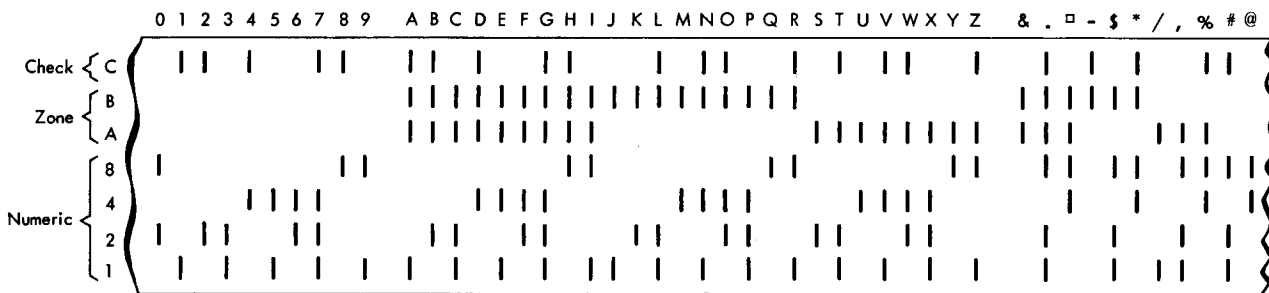


Figure 113. Magnetic Tape – Seven-Bit Alphameric Code

characters, peculiar to only one system, are written on tape. For this reason, consideration must be given to the characters used when tape written on one system may be used on another.

### Binary System

Binary information recorded on tape is related primarily to the IBM 704, 709, 7040, 7044, 7090, and 7094 Data Processing Systems. With these systems the basic unit of information is the word – 36 consecutive bits – compared to the character or digit of other systems.

To record a word of data on tape, the seven bit positions of each column on tape are used; however, the C bit position of the column is for parity checking only and is not considered a part of the word. Thus, six bits of information can be recorded in each column. A word of 36 bits is represented in six consecutive columns on tape (Figure 114).

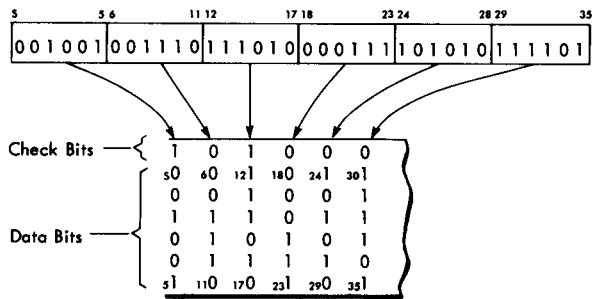


Figure 114. Magnetic Tape – Binary System

To verify accuracy of tape reading and writing, each column of bits must consist of an odd number of bits and is tested to insure odd parity. As tape is written, check bits are automatically added to the columns that have an even number of bits. In addition to this vertical parity check, a horizontal (longitudinal) parity check is made on each record. At the time a record is written, the bits of each horizontal row are counted. At the end of the record, a check character is recorded. This character has a bit corresponding to each row with an odd bit count. When the record is read, each row of the completed record, including the check character, should satisfy the even parity condition.

### Tape Records, Inter-Record Gaps, and End-of-File Gap

Records on tape are not restricted to any fixed length of characters, fields, words, or blocks. Records may be of any practical size within the limits of available storage capacity.

Records or groups of data are separated on tape by a record gap – a length of blank tape about 3/4 inch long. During writing, the gap is automatically produced at the end of the record. During reading, the record begins with the first data sensed after a gap and continues until the next gap is reached. The blank section also allows for starting and stopping the tape between records. A single unit or block of information is, therefore, defined or marked by an inter-record gap before and after the data (Figure 115).

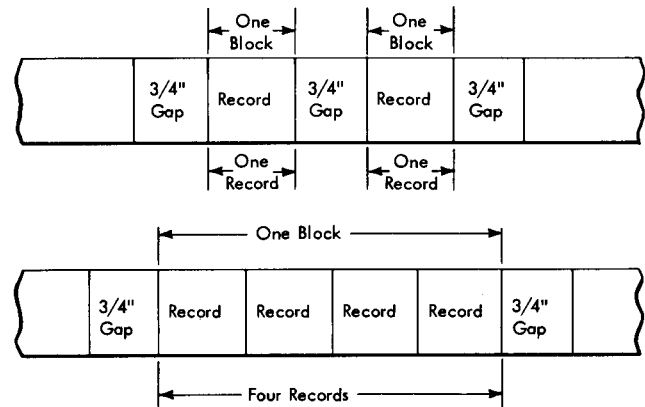


Figure 115. Single and Multiple Record Blocks

An inter-record gap, followed by a special single-character record, is used to mark the end of a file of information. The character, a tape mark (Figure 116), is generated and written on the tape following the last record of the file.



Figure 116. Tape Mark at End of File

More than one file may be placed on a tape, provided these files are separated by the end-of-file characters (tape marks). This is shown in Figure 117, where three files of varying numbers of records are recorded on tape.

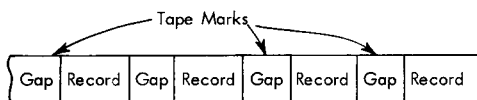


Figure 117. Multiple Files on a Tape

## Instructions

Input/output instructions select and control input/output operations. They contain information necessary to:

1. Identify the device or channel adapter required and the data channel to which it is attached.
2. Determine if the operation transmits data to core storage (read) or takes data from core storage (write).
3. Select appropriate code translators for the serial-by-character input/output devices.
4. Prepare the channel to accept a channel command word, which is sent to the data channel by execution of a reset and load channel instruction. The command is then executed by the data channel itself.
5. Prepare the data channel to accept a control word that contains an order for the 1301 Disk Storage. The control word is sent to the channel by the reset and load channel instruction and is then passed to the file control for actual execution.
6. Start mechanical motion in the input/output devices.

Thus, three different levels of execution exist in the 7040 and 7044 systems:

1. Instructions are executed in the processing unit.
2. Commands are executed in the data channel.
3. Orders are executed in the file control unit for disk storage.

Figure 118 shows the span of control and activity for execution of instructions, commands, and orders, together with data and information flow paths. Logically, data flow is the same with either type of data channel in the 7040 and 7044 systems. Actually, with

7904 Data Channels, data flow is a direct path between core storage and the channel; with Channel A (which is physically a part of the processing unit), data flow is as shown in the figure. Status data are available from all devices and are directed to core storage. Trap signals, also available from all devices, are directed to the processing unit but cause information to be placed in core storage at preassigned locations. Both status data and trap signals are discussed later. In Figure 118:

1. Instructions are taken from core storage and executed in the processing unit.
2. A command is taken from storage by an instruction, and is sent to the data channel for execution.
3. An order is taken from storage by an instruction and is sent to the file control for execution.
4. Status data, available from all devices, are sent to core storage. The data may then be taken from storage and interrogated by the processing unit to find the operating status of the device.
5. Trap signals, available from all devices and certain processing unit operations, are used to automatically force the program to a preassigned routine. This routine is then used to fix up the condition causing the trap signal and then to return program control to the area it was operating in when trapped. Detailed information on trapping is in the Trapping section.

## Special Condition Program Indicators

Each input/output device has special condition indicators used to signal the processing unit when a condition requiring attention occurs. Device indicators include: tape check on a tape unit, hole count on a card reader,

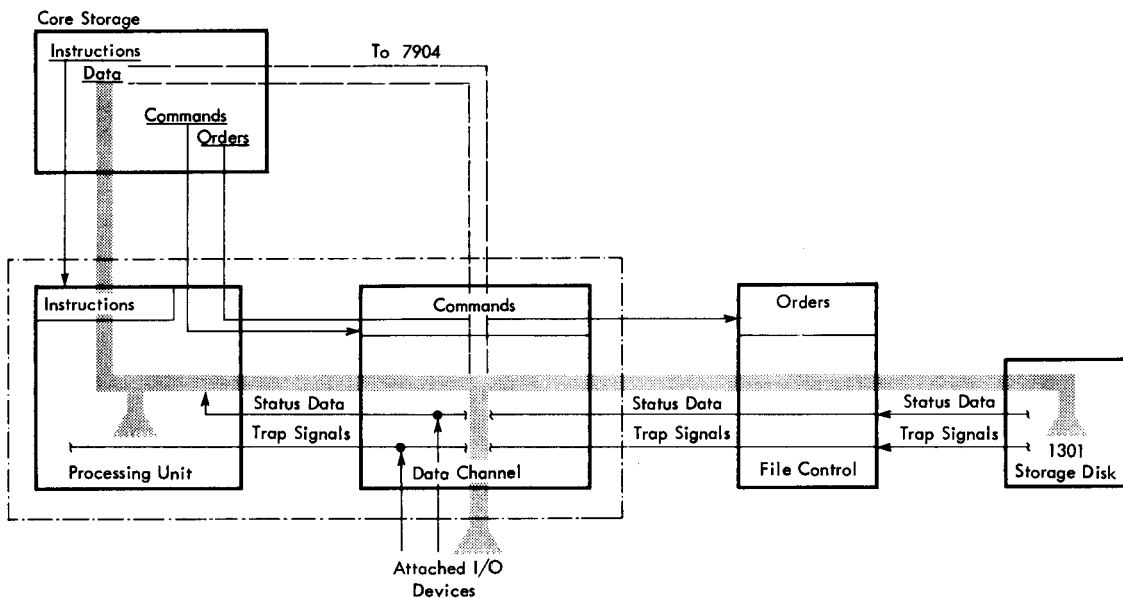


Figure 118. Instruction, Command, Order, and Data Flow

print echo on a printer, etc. In addition to this type of indicator, each data channel has indicators that are shared by all input/output devices attached to the data channel. Such an indicator might be the redundancy check indicator, which could be turned on by a tape check, a hole count, a print echo, or other causes to indicate that an error was sensed during data transmission between the input/output device and data channel.

The condition of the channel indicators may be program tested by instructions. Alternate program paths may be assigned to the results of the test. For example, if a channel redundancy check indicator were tested and found in the on condition (during a tape read operation), the alternate path might be to backspace tape and try to read the record again. Indicator descriptions are interspersed with instruction descriptions and examples in the following text.

In the following instruction descriptions, Y signifies the address part of the instruction (positions 21-35). This field selects the data channel and device to be used; see Figure 119. If a device is selected on channel A, additional information contained in positions 15-17 (designated V) is needed. Positions 15-17 are not interpreted by the 7904 channels (channels B through E). If the 1402 card punch is selected for use on channel A, additional information about stacker selection is necessary. This information is in position 13 and is also

designated by V. Thus, the V field includes positions 13-17. Examples are given in text where this field applies.

**Read Select — RDS Y,T,V**

This instruction causes the channel to prepare to read information into core storage from the input/output device specified by V and Y. Only positions 28-35 of the address part are subject to address modification by an index register. Position 14 of V must be a zero.

As an example of RDS coding, assume that tape unit 4 attached to channel A is selected for reading in the BCD mode. The format of this RDS instruction is:

RDS 644, T, 0

**Write Select — WRS Y,T,V**

This instruction causes the channel to prepare to write information from storage to the input/output device specified by V and Y. Only positions 28-35 of the Y field are subject to modification by indexing. Position 14 of V must contain a zero.

After a RDS or WRS has conditioned the channel to transmit data to or from a device, a reset and load channel (RCH) instruction must be given to deliver a channel command word to the channel. This command word contains the starting address of the data and a word count to control the number of words transmitted.

Device	Chan	Chan A Adapter	BCD Address		Binary Address	
			(Octal)	(Decimal)	(Octal)	(Decimal)
Magnetic Tape	A		1201-1212	0641-0650	1221-1232	0657-0666
	B		2201-2212	1153-1162	2221-2232	1169-1178
	C		3201-3212	1665-1674	3221-3232	1681-1690
	D		4201-4212	2177-2186	4221-4232	2193-2202
	E		5201-5212	2689-2698	5221-5232	2705-2714
7904 Control Adapter	B		2000	1024	2020	1040
	C		3000	1536	3020	1552
	D		4000	2048	4020	2064
	E		5000	2560	5020	2576
Direct Data Connection	B		2240	1184	2260	1200
	C		3240	1696	3260	1712
	D		4240	2208	4260	2224
	E		5240	2720	5260	2736
1622 Card Reader	A	3	1210	0648	1230	0664
1622 Card Punch	A	3	1211	0649	1231	0665
1402 Card Reader	A	3	1210	0648	1230	0664
1402 Card Punch	A	3	1211	0649	1231	0665
1403 Printer	A	3	1212	0650	1232	0666
Typewriter	A	4	01000	00512	01020	00528
1401 On-Line	A	5	1201-1212	0641-0650	1221-1232	0657-0666
1011 Paper Tape via 1414-4 or -5	A	3	1601	0897	1621	0913
1009 Data Trans via 1414-4 or -5	A	3	1301	0705	1321	0721
1014 Inquiry Units via 1414-4 or -5	A	3	1701-1702	0961-0962	1721-1722	0977-0978
Telegraph Units via 1414-4 or -5	A	3	1401-1404	0769-0772	1421-1424	0785-0788

Figure 119. Possible Select Instruction Addresses

### Reset and Load Channel A — RCHA Y,T

If the channel addressed by the RCH instruction has been prepared by a select instruction, the contents of the storage location specified by the Y field of the RCH are sent to the channel registers to serve as a channel command.

If the channel has not been prepared by a select instruction, the input/output check indicator in the channel is turned on.

The C(Y) are sent to the channel and data transmission starts as soon as the device selected is ready to operate. If a second RCH is given to a channel that is in operation (in use), the C(Y) specified by the second RCH are sent to the channel and replace the previous command. Since the second RCH resets the channel data register of the selected channel, data may be lost. One instruction exists for each channel; the mnemonics are: RCHA, RCHB, RCHC, RCHD, and RCHE.

Because timing conditions vary widely among data channels and their devices, it is recommended that the channel be tested to insure it is no longer in use before using the data area assigned to that channel. This test may be accomplished with a transfer on channel in operation instruction explained later.

### Channel Command — IORD Y,,V

Execution of the RCH instruction causes a channel command word to be sent to the addressed channel. This word provides a 15-bit word count field (specified by V) that regulates the length of the record (the number of words) moved. A 15-bit starting address field (specified by Y) is also provided in the command word to locate the first word to be moved. For example, if 450 words were to be read from a tape unit and the first word was to be placed in core storage location INPUT, the command word format would be:

IORD INPUT,,450

Additional words are taken from or sent to successively higher word locations in core storage until the end of record is reached on the input/output device,

or until the number of words specified in the word count (V) field has been transmitted; whichever of these conditions occurs first ends the operation. Thus, in the previous example, the second word taken from tape is placed in storage location INPUT+1, the third word in location INPUT+2, and so on. Each time a word is moved, the word count is automatically reduced by one. Likewise, each time a word is moved, the starting address is increased by one.

Figure 120 shows the RDS and RCH instructions with the IORD command. Assume that 250 BCD coded words are to be sent from tape unit 3, attached to channel B, into an INPUT core storage area. If the first record on tape were less than 250 words, only the number of words in that record would be sent to storage because an end of record is sensed by the tape unit at the end of the record, and this EOR signal takes precedence over the word count. If the first record were larger than 250 words, only the 250 words called for would be sent to storage, but the tape unit would continue until it reached the recorded end-of-record gap before it stops. The remainder of the record would be, in effect, skipped over.

By changing the RDS in Figure 120 to a WRS instruction, exactly 250 words would be written from storage on tape unit 3 as a complete record. Likewise, by changing the address portion from 1155 to 1171, the 250 words would be written in the binary mode (See Figure 119 for addressing).

### Channel-in-Use Indicator

This indicator is turned on by execution of any select instruction specifying that channel. Normally, the indicator remains on during the reset and load channel instruction, which loads the channel with a command word. The indicator stays on during execution of the command and is turned off at completion of the command.

Since the select instruction initiates mechanical motion in the tape unit, the RCH instruction must be

Location		Operation	Address, Tag, Decrement/Count	Comments	Identification		
1	2	4	7	8	72	73	80
		RDS	1155	Selects tape unit 3 on channel B.			
		RCHB	I.COM	Brings channel command into the control registers of channel B.			
		I.COM	BCD	INPUT,,250	The command counts 250 words from the first record on tape and places these words into storage locations INPUT, INPUT + 1, INPUT + 2, etc.		

Figure 120. Tape Record Read Routine

executed within a certain time after the select instruction. This time depends on the tape unit being used and is noted in milliseconds as:

UNIT	READ	WRITE
7330	7.0	13.0
729 II	4.0	6.5
729 IV	2.5	4.0
729 V	4.0	6.5
729 VI	2.5	4.0

If the RCH is not executed within the allowed time, the tape unit is disconnected and the input/output check indicator is turned on.

The channel-in-use indicator is also turned on for the duration of the specified operation. If a select, backspace, write end of file, rewind, rewind and unload, or write blank tape instruction is given with the channel-in-use indicator on (as a result of a previous instruction), the execution of the instruction is delayed until the channel-in-use indicator is turned off.

The channel should not be in operation when selected for use. The channel status may be tested and, if the channel is busy, a program delay (TCO instruction) may be used. Any other scheme that brings the program back to testing the channel after doing other operations may also be used.

#### Transfer on Channel A in Operation — TCOA Y,T

If the channel-in-use indicator is on for the specified channel, the computer takes its next instruction from location Y. Otherwise, the next sequential instruction is taken. The operation of the channel is not affected by the TCO and the TCO is never treated as a no-operation. When used with channel A, the TCO, if it does not transfer, tells the program that the channel has not been loaded with a RCH instruction. One instruction exists for each data channel; the mnemonics are: TCOA, TCOB, TCO C, TCO D, and TCO E.

The TCO instruction, in its simplest application, may be given an address that is the same as its location. With this application, the TCO could be inserted in the program instruction string immediately in front of the RDS or WRS (Figure 121). If the channel-in-use indicator is on for channel B, the next instruction is taken from location START. This one instruction loop continues until the channel-in-use indicator is not on.

The next sequential instruction (RDS or WRS) is then executed.

The main limitation to this type of TCO use is that the program is held in the one instruction loop until the channel is not busy. The alternative method of testing the channel and transferring to other work if the channel is busy saves over-all program time but might require more instructions. With this use, the address of the TCO would be the location of other work to be done. The last instruction in the other work routine would have to be a transfer back to the TCO to find out if the channel is now ready for additional input/output work.

#### Redundancy Check Indicator

Another aspect of data movement (reading or writing) is the possibility of a parity error indication. This tells the computer that the check bit with the character (or word) does not agree with the number of bits within the character (or word).

The redundancy check indicator, one for each data channel, may be turned on at any time during a read or write operation. It is turned on when a parity error is detected on data being moved to or from core storage. The indicator is turned off by execution of a transfer on redundancy check instruction.

#### Transfer on Redundancy Check, Channel A — TRCA Y,T

If the redundancy check indicator for the specified channel is on, it is turned off and the computer takes its next instruction from the location specified by Y. If the indicator is off, the next sequential instruction is taken. One instruction exists for each data channel; the mnemonics are: TRCA, TRCB, TRCC, TRCD, and TRCE.

Since it is possible for the redundancy check indicator to be on from a previous operation, the indicator should be turned off (if on) before a read or write operation is started. This may be done by placing the TRC directly in front of the RDS or WRS in the instruction string. The TRC is also placed in the program after the RCH to check data movement as it occurs (Figure 122).

The first TRC of Figure 122 would turn off the redundancy check indicator if it were on and then allow

Location	Operation	Address, Tag, Decrement/Count	Comments	Identification
1 2 4 7 8				72 73 80
START	TCO B	START	The TCO loops until the in use indicator is off.	
	RDS or WRS		Select for reading or writing.	

Figure 121. TCO Instruction Loop

execution of the RDS, RCH, and IORD. The second TCO is used to insure that the following TRC does not attempt any testing before the data are actually in computer core storage. Without the second TCO instruction, the second TRC would be executed before any data had been read by the tape unit. This points up the difference between time needed to get a mechanical device moving and reading data and the time needed to execute computer instructions. Approximately 200 computer instructions could be executed before data would start to arrive from the tape unit.

The OUT address of the second TRC is the location of a re-read routine that would be needed if error data were encountered. The OUT routine would basically consist of instructions to backspace the tape record and to re-read it. Present programming practices suggest ten read attempts before the record is considered unreadable.

**Backspace Record – BSR Y,T,V**

This instruction causes the tape unit designated by Y and V to move tape backward. This backward motion is continued until an end-of-record gap or load-point gap is reached. If the tape unit is at load point when this instruction is executed, the BSR is treated as a no-operation and the next sequential instruction is taken. Only positions 28-35 of the address part of the BSR are subject to address modification.

There are five tape areas where the tape can stop when operating under normal conditions. When the tape is stopped, the read head is positioned at one of these areas. The areas are labeled 1 through 5 on Figure 123. Operation of BSR and RDS instructions varies, depending on tape positioning when the instruction is executed (Figure 124).

Tape is positioned at the load-point marker (1) by execution of a rewind instruction or by the initial mechanical loading of a tape reel and depression of

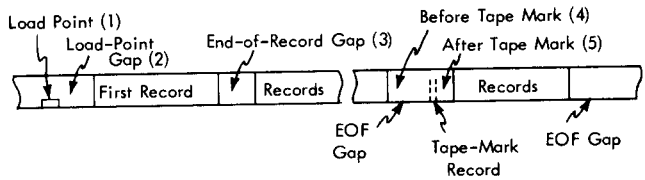


Figure 123. Stopping Areas on Magnetic Tape

the tape load-rewind key. Positioning in the load-point gap (2) occurs when a BSR is executed after the first record has been read or written on the tape.

Tape is stopped in an end-of-record gap (3) when a record is read or a BSR is executed at the end of any record other than the first record of a file. Tape is stopped before the tape mark (4) after the last record of a preceding file is read or after a BSR is executed for a tape positioned after the tape mark. Tape is positioned after the tape mark (5) by execution of a BSR for a tape positioned after the first record of the next file, or by execution of a RDS for a tape positioned before the tape mark.

Figure 124 shows the differences in instruction execution according to tape position; the numbered positions are the same as in Figure 123.

With Tape Positioned	(1) At Load Point	(2) In Load-Point Gap	(3) In EOR Gap	(4) Before Tape Mark	(5) After Tape Mark
BSR	No-operation. Take next instr.	Back tape to load point	Back tape over previous record of this file, if one exists.		
RDS	Read first record of the first file		Read next record of this file	Read an EOF condition	Read first record of next file.

Figure 124. Tape Instruction Execution

Location	Operation	Address, Tag, Decrement/Count	Comments	Identification
1 2 6 7 8	START	T C O B	START	72 73 80
			Check for channel busy.	
			MOVE	
			Turn off redundancy check, if on.	
			XXXX	
			Select channel, unit, and mode.	
			ICON	
			Load command into the channel.	
			CHECK	
			Holds program until data have	
			actually been read.	
			OUT	
			OUT is the location of a re-read	
			routine for the error condition.	
			Channel command.	
			ICON	

Figure 122. TRC Instruction Example

Figure 125 shows a basic routine for re-reading an error record a maximum of ten times. Transfer to this OUT routine from the one shown in Figure 122 occurs when the second TRC finds an error in data transmission.

The AXT places 9 into the XR to serve as a re-read counter (the record has already been read once). The tape unit is then tested by the first TCO and, when the BSR is complete, the record is read (with the RDS, RCHA, and IORD sequence) a second time. The next TCO checks for "not busy" and the TRC checks this second reading. The TRA instruction following the TRC is an unconditional transfer back to the main program to continue processing. If the record is again read as an error record, the TRC transfers to location COUNT. At COUNT, the contents of XRI are reduced by one and the program returns to the backspace instruction to try re-reading a third time. This process continues until the record is read as good, or until the XR is reduced to 1(TIX). At this time, the record has been read ten times as an error and may be considered bad. Some sort of operator intervention is needed, or the program could type out that record number *x* could not be read and continue without stopping.

For writing on tape, similar routines would be used (WRS instead of RDS) and, after twenty-five tries to write a good record, that area of tape is assumed bad. Instead of operator intervention being indicated, a blank section of about 3½ inches of tape is written.

### Write Blank Tape — WBT Y,T,V

This instruction causes the tape unit designated by Y and V of the WBT to write a blank area of tape about 3½ inches long. The instruction is used to erase bad spots on tape which cannot be written over without a redundancy check indication. The WBT must have a

1 bit in position 14 and should not be immediately followed by a RCH instruction.

To illustrate use of the write blank tape instruction, assume a tape write routine (similar to Figure 122) and a TRC to the OUT routine, signaling a write error. The OUT routine to attempt rewriting the record could be as shown in Figure 126. The AXT places 25 in the XR and the tape unit is backspaced one record. The TCO holds the program until the BSR is complete, and the WRS is then executed, along with the RCH and IORD. Again a TCO is used, followed by a TRC. If the record is written without error this second time, program control is returned to the main program for continued processing. With a second error indication, the TRC transfers to COUNT. COUNT reduces the counter by 1 and transfers back to the BSR instruction. This procedure continues until the record is written as a good record or the count is reduced to 1. At this time, the WBT instruction is executed, effectively skipping over the suspected bad spot on tape. The rewrite routine then transfers back to the original WRS instruction in the main program and attempts to write the record on the new section of tape.

Note that a count (30) of possible skips is kept by index register 2. If this count is exceeded on one record writing attempt, the operator should change tape units and notify the Customer Engineer.

### Tape Marks

When the series of tape records making a tape file has been successfully written, this series must be marked to make it a recognizable tape file. This is done by writing a single-character record called a tape-mark record. The tape mark consists of 1 bits in the 8, 4, 2, and 1 tracks and is followed by a check character for the tape mark.

Location				Operation	Address, Reg, Decrement/Count	Comments	Identification		
1	2	6	7	8			72	73	80
	OUT			AXT	9,1	Places 9 in XRI.			
	BACK			BSR	B3	Backspaces the tape one record.			
	RDY			TCOB	RDY	Waits until backspace is finished.			
				RDS	B3	Selects same tape for re-reading.			
				RCHB	ICON	Load channel command.			
	SELF			TCOB	SELF	Holds program until data is ready.			
				TRCB	COUNT	Error, go to COUNT.			
				TRA	NEXT	Good, return to main program.			
	COUNT			TIX	BACK,1,1	Reduce counter, transfer to BACK.			
				HPR	OUT	Record cannot be read without error. If another			
						ten tries are desired, press the start key.			
	ICON			BCI	IN,17	Channel command.			

Figure 125. Re-read Routine



### End-of-File Indicator

The channel end-of-file indicator is turned on only when a single-character tape-mark record is sensed during a read operation. An end-of-file record may be written on a tape by the write-end-of-file instruction, but the end-of-file indicator is not turned on during this operation. The indicator may be tested by the transfer-on-end-of-file instruction and, if the indicator is on when tested, it is turned off by execution of the test instruction.

### Write End of File — WEF Y,T,V

This instruction causes the tape unit designated by Y and V to write an end-of-file gap followed by a tape-mark character on the tape. If the end-of-tape reflective marker is passed over during the WEF instruction, the end-of-tape indicator is turned on. Only positions 28-35 of the WEF are subject to address modification.

### End-of-Tape Indicator

When the end-of-reel marker is sensed during writing, the end-of-tape indicator in the channel to which the tape unit is attached is turned on. No interruption in the writing process occurs, so that the write operation may be completed even though the end-of-reel marker has been passed over. If the status of the indicator is ignored and writing continues, the tape may be pulled from the reel. This indicator is never turned on during a read operation.

The end-of-tape indicator may be turned on during execution of a WRS, WEF, or WBT instruction. It is not turned on during a RDS. If it is turned on during a WRS, writing does not stop but continues until the end of the record or until the end of operation. Since, however, this record may not be the last record in a file, writing should stop on this reel of tape and restart on a new tape reel.

### Transfer on End of File, Channel A — TEFA Y,T

If the end-of-file indicator for the specified channel is on, it is turned off and the computer takes its next instruction from Y of the TEF instruction. If the indicator is off, the next sequential instruction is taken. The EOF indicator may be turned on by magnetic tape, a card reader, or the on-line IBM 1401 System.

When an end of file is sensed during reading, the turning on of the channel end-of-file indicator logically disconnects the tape unit from the channel. Execution of the command word is terminated immediately, even if it has not been completed. One instruction exists for each channel; the mnemonics are: TEFA, TEFB, TEFC, TEFD, and TEFE.

The recognition of an end of file occurs with execution of a RDS instruction when the tape mark is spaced over. Assuming that records and a tape mark have been written on a tape, a routine to sense the end of file could be as shown in Figure 127. Records are read in a normal manner and tested for validity. When the EOF is sensed, the TCOA is executed until the channel is not in use; then the TEFA is executed.

Location		Operation	Address, Tag, Decrement/Count	Comments	Identification
1	2	6	7	8	72, 73
	OUT	AXT	9,1	9 to XR1.	
	BACK	BSR	B3	Backspace one record.	
	RDY	TCOB	RDY	Backspace finished.	
		WRS	B3	Try writing once more.	
		RCHB	ICON		
	SELF	TCOB	SELF	Ready for checking.	
		TRCB	COUNT	Error, go to COUNT.	
		TRA	NEXT	Good writing, return to main program.	
	COUNT	TIX	BACK,1,1	Reduce write counter, transfer to BSR.	
		BSR	B3	Backspace for WBT position.	
		WBT	B3	Write blank tape.	
		AXT	29,2	29 to XR2.	
		TIX	MAIN,2,1	Reduce skip counter, transfer to main program.	
		HPR		Change tape unit.	

Figure 126. Rewrite and Skip Routine

**End of Tape Test, Channel A — ETTA Y,T**

This instruction tests the status of the channel end-of-tape indicator. The channel whose indicator is to be tested is specified by Y of the **ETT** instruction. If the end-of-tape indicator for channel Y is on, the computer takes the next sequential instruction and turns the indicator off. If the indicator is off, the computer skips the next instruction and proceeds from there. One instruction exists for each channel; their mnemonics are: **ETTA**, **ETTB**, **ETTC**, **ETTD**, and **ETTE**.

The instruction combination shown in Figure 128 would determine whether the end-of-tape marker has been passed during a write operation.

If the end-of-tape marker is encountered while the tape unit is being stopped, the channel end-of-tape indicator is turned on but is not recognized by the program until a succeeding (another) **ETT** instruction is executed. This situation could occur when the end-of-tape marker falls in an end-of-record gap.

**Rewind Instructions**

Two instructions, available for each channel, will rewind the tape reel to its load point. One simply rewinds the tape reel; the other rewinds the tape and then unloads the reel for physical removal from the tape unit.

**Rewind — REW Y,T,V**

This instruction causes the tape unit designated by Y and V to rewind its tape to the load-point position. If the tape is positioned at its load point when the **REW** is executed, the instruction is treated as a no-operation. Only positions 28-35 of the address part of the **REW** are subject to address modification. On a 7330 tape unit, this instruction causes a low-speed rewind.

**Rewind and Unload — RUN Y,T,V**

This instruction causes the tape unit designated by Y and V to rewind its tape and then to unload that tape reel. The **RUN** is treated in the same manner as the **REW** except, after the rewind operation, a normal unload (manual) operation occurs. On a 7330 tape unit at load point, the **RUN** causes the channel to halt operation. With 729 tape units at load point, the **RUN** causes an unload operation only.

**Input/Output Check Indicator**

If the data channel being used has not been conditioned by a select instruction when the **RCH** instruction is executed, the input/output check indicator is turned on (one indicator for all channels). This condition may be tested by using the **IoT** instruction.

The indicator is turned on by any of the following conditions:

Location		Operation	Address, Tag, Decrement/Count	Comments	Identification
1	2	4	7	8	72 73 80
		<b>RDS</b>	<b>B3</b>		
		<b>RCHB</b>	<b>ICON</b>	Get channel command and load channel.	
<b>SELF</b>		<b>TCOB</b>	<b>SELF</b>	Holds program until channel is free.	
		<b>TEFB</b>	<b>CHANGE</b>	EOF is sensed.	
		Continue			
<b>ICON</b>		<b>BCI</b>	<b>IN,,n</b>		

Figure 127. TEF Program Example

Location		Operation	Address, Tag, Decrement/Count	Comments	Identification
1	2	4	7	8	72 73 80
		<b>WRS</b>	<b>B3</b>	Select tape and channel.	
		<b>RCHB</b>	<b>OCON</b>	Get output command.	
<b>SELF</b>		<b>TCOB</b>	<b>SELF</b>	Hold program until write is finished.	
		<b>ETTB</b>	<b>1000</b>	Test channel indicator.	
		<b>REW</b>	<b>B3</b>	Indicator on.	
		Continue		Indicator off.	
<b>OCON</b>		<b>BCI</b>	<b>OUT,,n</b>		

Figure 128. ETT Program Example

1. If a RCH is decoded and the data channel has not been selected for use.

2. If, during writing, a channel data register has not been loaded with a word from storage by the time its contents are to be sent to the output device.

3. If, during reading, a channel data register has not sent its contents to storage by the time new data are to be loaded into the data register from an input device.

The indicator is turned off by execution of an input/output check test (IOT) instruction.

#### Input/Output Check Test — IOT, T

If the input/output check indicator is on, the indicator is turned off and the computer takes the next sequential instruction. If the indicator is off, the computer skips the next instruction and proceeds from there. Any address modification may result in changing the operation because the Y portion of the IOT is a part of the operation code itself.

#### Channel Reset and Store Instructions

The contents of data channel registers may be reset to a starting condition or their contents may be stored in core storage for use by the program.

#### Reset Data Channel A — RDCA, T

This instruction resets all registers and indicators in the data channel specified by the RDC instruction. All data transmission is terminated, and the selected devices are immediately disconnected. If the instruction is executed while a tape is in motion, the tape is immediately stopped, regardless of the position of the tape head with respect to the inter-record gap. All status indicators previously set by an enable instruction (explained later) are turned off. A RDC cancels the effect of a previous select instruction. One instruction exists for each channel; the mnemonics are: RDCA, RDCB, RDCC, RDCE, and RDCD.

#### Store Channel A — SCHA, Y, T

This instruction replaces the C(Y)<sub>21-35</sub> with the contents of the specified channel's address counter. Positions 3-17 are replaced with the contents of the channel's word counter. Positions S,1,2,18,19, and 20 are set to zero. Since channel A uses the C(AC)<sub>3-17</sub> for its word counter and C(AC)<sub>21-35</sub> for its address counter, it is necessary to give the SCHA before changing the AC after execution of an RCHA.

An SCH to channels B through E may be executed at any time, regardless of whether the specified channel is in operation. If the channel is in operation and the channel's address counter is in the process of being changed, execution of the SCH is delayed until the

change is complete. The contents of the address counter are one greater than the storage location of the last word involved in the data transmission. One instruction exists for each channel; the mnemonics are: SCHA, SCHB, SCHC, SCHD, and SCHE.

#### Problem

34. Write a routine to read a 15-word binary record from tape unit 3 attached to data channel B. Delay checking until the record is read into core storage, and then check for redundancy and end of file. If redundancy occurs, try to re-read ten times. If redundancy persists, halt with 77777<sub>8</sub> in the address portion of the storage register. If end of file occurs, halt with 11111<sub>8</sub> in the address portion of the storage register. For normal end, halt with 00001<sub>8</sub> in the address portion of the storage register.

#### IBM 1414-3, -4, and -5 Input/Output Synchronizers

These models of the 1414 provide data buffer storage and control functions for the following units:

IBM 1414-3	IBM 1403 Printer IBM 1402 Card Read Punch
IBM 1414-4	IBM 1402 Card Read Punch IBM 1403 Printer IBM 1402 Column Binary Adapter. Two buffers are required (one adapter per 1414). IBM 1009 Data Transmission Unit Adapter. One adapter controls one IBM 1009. Two buffers are required (one adapter per 1414). IBM 1011 Paper Tape Reader Adapter. One adapter controls one IBM 1011. One buffer is required (one adapter per 1414). IBM 1014 Remote Inquiry Unit Adapter. One adapter controls up to ten IBM 1014's. Two buffers (one for input and one for output) are required for each adapter (maximum of two adapters per 1414). Telegraph Input/Output Feature. One adapter attaches two simplex circuits or one half-duplex or full-duplex circuit. Two buffers are required for each adapter (one adapter per 1414). Additional Telegraph Input Feature. One adapter attaches one simplex, half-duplex, or full-duplex circuit in conjunction with the additional telegraph output feature. One buffer is required for each adapter (maximum of two adapters per 1414). Additional Telegraph Output Feature. One adapter attaches one simplex, half-duplex, or full-duplex circuit in conjunction with the additional telegraph input feature. One buffer is required for each adapter (maximum of two adapters per 1414).
IBM 1414-5	Only the communication-oriented input/output devices used on the 1414-4 are available on the 1414-5.

With communication-oriented devices, any combination of the optional adapters is permitted, provided

that the limitation on multiples of the same adapter and the limit of six data buffers per 1414 are not exceeded.

### **Input Operation**

Normal input operation from any input buffer uses an RDS or PRD instruction followed by an RCHA instruction that loads a channel command into the channel registers. The select instruction selects the proper input buffer, but no action occurs until the RCHA is given. Therefore, no time limit exists between the select and the RCHA instructions.

The select instruction also samples the check status of the input buffer. If a check exists on the record in the buffer, the channel A redundancy check indicator is turned on. During the RCHA, the characters are also checked for parity as they enter the channel and, if improper parity exists, the channel A redundancy check indicator is turned on.

### **Output Operation**

Normal output operation to any of the output buffers uses a WRS or PWR instruction followed by a RCHA that loads a command into the channel. The select instruction selects the proper output buffer, but no action occurs until the next RCHA. Therefore, no time limit exists between the select and the RCHA. The select also samples the status of the output buffer. Normally, this is the status of the previous record; in a card punch operation, it is the status of the card punched before the previous card. If a check occurs on this record, the channel A redundancy check indicator is turned on.

During the output data transfer, the 1414 checks for proper parity on the data sent by the CPU. If an error occurs, the channel A redundancy check indicator is turned on. If the redundancy check indicator is on at the end of an output data transfer, the output buffer is not emptied and the data transfer from the 1414 is effectively cancelled.

## Punched Cards, Readers, Punches, and Printers

In most applications, magnetic tape is the principal input medium. It may be desirable to use IBM cards as input in some situations where the volume of input is small enough to permit economical operation. In either case, IBM cards are used as the medium for initially recording data because of their great flexibility: errors are easily detected and corrected, input data may be readily prepared on several card-punches simultaneously, and the cards may be collected before entry into the computer.

Cards are particularly useful when manual access to a file is desired. Punched card input and output may represent any alphabetic character, special symbol, or binary punching if the programs which manipulate this information are designed to recognize the code used (Figure 129). Input card information is normally coded in one of two ways: IBM card coding (Figure 129) or column binary.

### Column Binary Feature

This optional feature permits reading or punching of column binary cards or IBM card coded cards intermixed on the IBM 1402 Card Read Punch. The feature

is not available on the IBM 1622 Card Read Punch. A card recorded in column binary is identified by a 7 and 9 punch in card column 1. The 7-9 punches are sensed at the read check station of the 1402 reader. The card is then read at the read station so that card rows 12-3 are read into one buffer and rows 4-9 are read into another buffer.

The first character (six bits) is punched in card rows 12-3 of card column 1. The second character (7 and 9 control punches) is placed in card rows 4 through 9 of card column 1. The third character is punched in rows 12-3 of card column 2, and so on through card column 80. When all card columns have been used, 160 characters may be recorded on a single card instead of the 80 characters possible with IBM card coding. Figure 130 shows a card partially recorded in column binary format. With column binary recording, as with IBM card coding, the computer regards any punched hole as a binary 1. No punch indicates a binary 0.

The first character recorded on a column binary card is normally used for instruction count, etc., for that card. The second character is always a 7 and 9 punch

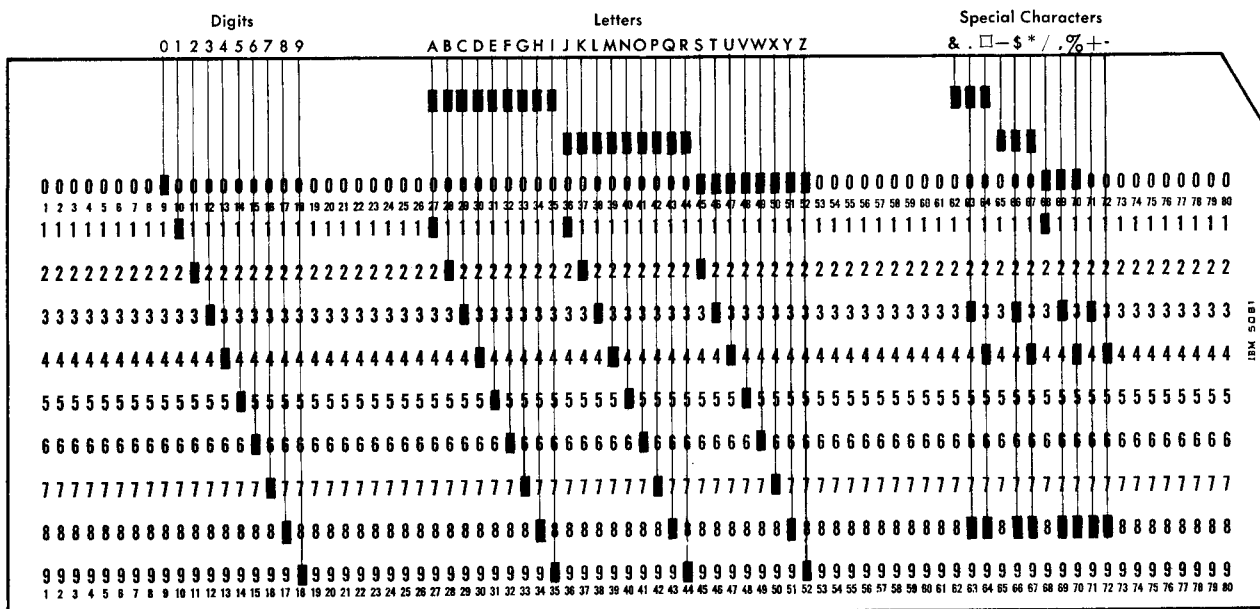


Figure 129. Standard IBM Card

(the six-bit number 000 101). The third character, rows 12-3 of card column 2, is usually the first data character; rows 4-9 of card column 2 contain the second character, etc. Each character is a six-bit binary number and each requires six card rows for recording.

As an example of column binary recording (Figure 130) assume that the first card character is used for a check sum, the second character must have a punch in the 7 and 9 rows; the third character then is the first data character. The data word (six characters) recorded on the card is:

-00101 101110 001000 110111 010110 011101 or  
 -055610672635.

For reading and punching cards, a record is defined as the information contained in one card. A file consists of any number of cards (records) and takes the form of a deck of cards. Note here that definitions of records and files depend on the device being used; for example, a record on magnetic tape may contain more than one card record.

### Card Stackers

Two card stackers are provided for each feed unit of a card reader: one for normal stacking and the other for error selected stacking. The physical stackers on the 1402 and 1622 readers are identical and are shown in Figure 131, along with their use with each device.

### IBM 1622 Card Read Punch

This unit operates at 250 cards per minute while reading and 125 cards per minute while punching (writing). The read and punch portions are separate and functionally independent, with separate switches, lights, checking circuits, and buffer storage.

Cards are fed face down, 9-edge first, and an entire 80-column row is read at a time. This row, and the eleven rows following it, are placed in a buffer storage and held there until all information from that card is in buffer storage. The reverse is true when cards are being punched; cards are punched face down, 12-edge first.

Punch Feed							Read Feed	
		NP	4	8/2	1	NR		
Pocket		1402 Use			1622 Use			
NP		Error Punch			Normal Punch			
4		Normal Punch			Error Punch			
8/2		Selected Punch			Not Used			
1		Normal Read			Error Read			
NR		Error Read			Normal Read			

Figure 131. Stacker Pockets on 1402 and 1622

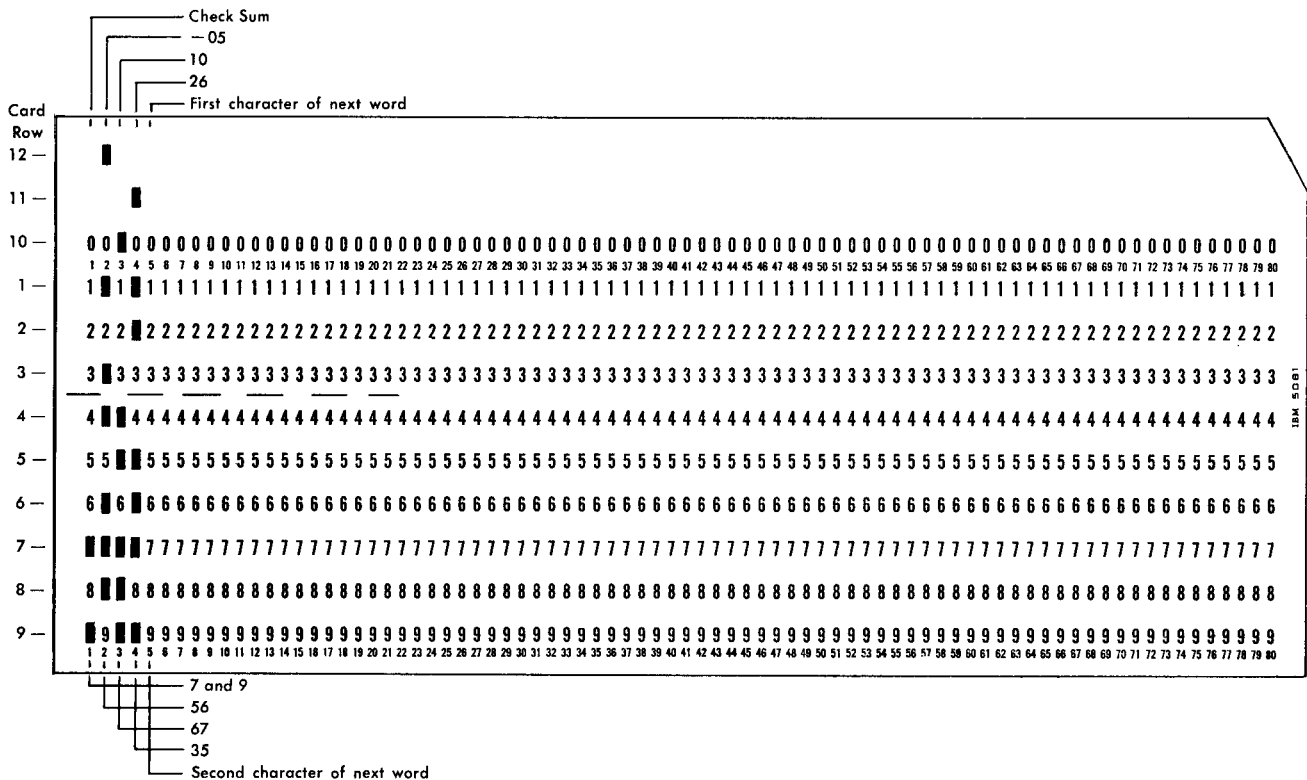


Figure 130. Column Binary Recording

When the 1622 is attached to channel A, the 1414-3 or -4 Input/Output Synchronizers may not be used on channel A. Likewise, with a 1414-3 or -4 attached to channel A, the 7040 or 7044 system cannot have a 1622.

### **IBM 1402 Card Read Punch**

This unit operates at 800 cards per minute while reading and 250 cards per minute while punching. As with the 1622, the read and punch portions are separate and functionally independent.

The 1402 is attached to the computer system through the 1414-3 or -4 Input/Output Synchronizers. The 1402 has its own buffer storage within the 1414-3 or -4, thereby holding up the processing unit only for the length of time required to fill or to empty the buffers.

Cards are read face down, 9-edge first, and punched face down, 12-edge first. The entire 80 columns of the card may be read or punched.

### **Card Reader Operation**

The read buffer is initially filled when cards are fed into the reader by the operator. Whenever a read operation is executed, the entire contents of the buffer are read out and a card feed cycle refills the buffer with the contents of the next card. The actual storing of data is under control of the channel `IORD` command. One command is required for each card read, but up to 80 card columns may be read from the card.

With IBM card code recording, 13 complete word locations (six characters each) and the 12 high-order (S, 1-11) positions of a 14th location are used to contain the 80 character positions on a card. Positions 12-35 of the 14th location are filled with binary zeros.

With column binary recording (1402 only), 26 complete word locations and the 24 high-order (S, 1-23) positions of the 27th location are used to contain the 160 character positions on a column binary card. Positions 24-35 of the 27th location are filled with binary zeros.

For a normal operation, the program uses a read select instruction followed by a `RCH`, which loads the `IORD` command into the channel registers. To read a full card, the `IORD` must have a word count of 14 or greater for IBM card code cards or 27 or greater for column binary cards. Word counts greater than 14 or 27 are treated as 14 or 27.

### **Read Operation — 1622**

Cards are read 9-edge first, face down, past two reading stations: check and read. The read buffer is initially loaded with 80 columns of card data during a start or load run-in operation. Thereafter, each card

feed cycle is under program control. The reader can accept and translate card codes equivalent to the 64 combinations of six bits (with optional feature).

The channel transfers data to storage until 80 characters are read or until the word count is reduced to zero, whichever occurs first. This results in an efficient read operation, because reading one card per command allows the CPU to process while the mechanical card reading process is taking place.

A read select directed to the 1622 causes a read signal to be sent to the reader. If the read buffer is not ready, the read signal is delayed. On receipt of the signal, the 1622 takes a read buffer cycle and transmits one data byte. A service request is also sent to indicate the presence of the byte. The channel takes the byte into the MQ register and sends a response to the 1622. This response causes another read buffer cycle and another byte is transferred. This request and response process continues until the entire read buffer is emptied. The channel stops data transmission when the word count goes to zero but remains connected to the 1622 until the end-of-record signal occurs. The channel then ends operation, and the 1622 reads the next card into the read buffer. The previous card is stacked in the NR pocket unless an error has occurred.

### **Special Read Conditions — 1622**

1. Each card is read at two different places and the results of the readings are compared. An unequal comparison is called a hole check. If a hole check error is detected, the card feed stops, ready status is ended, and the reader check indicator is turned on. The card in error is placed in the error stacker. A transfer-on-device-in-operation directed to the reader results in a transfer, and the read select instruction causes the channel to halt operation. The channel redundancy indicator is not turned on and manual intervention is required. The error card is placed in pocket 1.

2. Each data byte is parity checked as it leaves the read buffer and, if a parity check is detected, the 1622 follows the same procedure as with the hole check. All conditions are the same, except the error card is placed in the normal stacker.

3. Each byte received by the channel is parity checked. If a parity check is detected, the channel redundancy indicator is turned on and the read operation continues to normal completion. The redundancy indicator should be checked by the program to assure that the transfer was valid.

4. An end-of-file signal is sent to the channel when the last card in the read feed has been transmitted. The signal turns on the channel end-of-file indicator when the next read select addressing the reader is given.

### **Write Operation — 1622**

Cards are fed 12-edge first, face down, past the punch and check stations. All 64 combinations of six bits (with optional feature) can be translated and punched. Information is transferred from storage until 80 characters are written or until the word count is reduced to zero. A write select instruction addressing the 1622 results in a service request for the first byte. The 1622 stores this byte in its punch buffer and then requests the next byte. This request and response process continues until the entire punch buffer is filled. When the word count goes to zero, the channel stops sending words but continues sending blanks to the 1622 until an end-of-record signal is received. The channel now ends operation and the 1622 proceeds to punch the data just transferred. The card is placed in the NR pocket unless a parity or punch check occurs.

### **Special Write Conditions — 1622**

1. The channel checks parity on all words sent from storage. If a parity error is detected, the channel word parity indicator is turned on and an error signal, which prevents the error record from being punched, is sent to the 1622.

2. The 1622 checks parity on all bytes received from the channel and on all bytes punched out of the punch buffer. If a parity error or a punch error is detected, a cycle delay is started and the punch is stopped one card feed cycle after punching the incorrect data. Ready status is terminated and the punch check light is turned on. A `TDOA` instruction directed to the punch will not transfer and a write select instruction fills the punch buffer, but no punching occurs. The next `TDOA` directed to the 1622 transfers, and a write select instruction will now halt operation. The channel redundancy check is not turned on and manual intervention is required to clear the condition. The error card is placed in pocket 4.

### **Read Operation — 1402**

The read buffer is initially filled when cards are fed into the reader by the operator. Whenever a read operation is executed, the entire contents of the buffer are read out and a card feed cycle refills the buffer with the contents of the next card. The actual storing of data is under control of the channel command. One command is required for each card read, but up to 80 characters may be read from the card. Since each core storage location can contain six characters, 13 complete word locations are used and the 12 high-order positions of a 14th location contain the 79th and 80th characters; low-order positions are filled with zeros.

For a normal operation, the program uses an `RDS` instruction followed by an `RCH`, which loads the `IORD`

command into channel registers. The `RDS` selects the read buffer, but no action is taken until the following `RCHA`. Hence, the time between the `RDS` and the `RCHA` is variable.

### **Special Read Conditions — 1402**

1. The 1414-3 recognizes 64 valid characters. Any card code that does not result in a valid character causes a reader validity error. Channel A redundancy check is turned on when the `RDS` is given for that card, and the card is placed in the NR pocket.

2. When a card passes the read check station, the number of holes in the card are counted. A hole count check occurs if the comparison is not equal, and the channel A redundancy check is turned on when the `RDS` is given for that card. The card is placed in the NR pocket.

3. Data read from the buffer are checked for proper parity by the CPU. If a parity error is detected, the redundancy check indicator is turned on. The program should test this indicator for the corrective action required.

4. A not-ready condition results from reader out of cards and not end of file, reader in manual status (off-line), or reader power off. These conditions require operator intervention. When a read select instruction is executed for the reader, the CPU halts operation.

5. If a hole count or parity error is detected, the channel redundancy check indicator is turned on and may be tested by the program.

6. If the buffer is being filled, a read buffer busy condition exists. If a read select is given, the CPU waits for a not-busy condition.

7. The end-of-file indicator in the 1414 is turned on after data from the last card have been sent to core storage. On the next select instruction to the reader, the end-of-file indicator in the CPU is turned on and the end-of-file indicator in the 1414 is turned off.

### **Write Operation — 1402**

The punch buffer has a capacity of 80 characters plus parity. Words are sent to the punch in the same way as with the reader, except that a write select instead of a read select instruction is used. When C words have been sent to the punch, blanks are inserted in unfilled buffer positions. When the buffer is full, a punch card feed cycle is initiated. The channel is then disconnected and the CPU executes the next sequential instruction. For a normal punch operation, the program uses a `WRS` followed by an `RCHA`, which loads the `IORD` command into the channel registers. The `WRS` selects the buffer, but no action occurs until the `RCHA` instruction is executed.



The program can select one of two pockets in the 1402 to stack the punched cards. If the write select instruction has a zero in position 13, the card is stacked in pocket 4. If the write select instruction has a one in position 13, the card is stacked in pocket 8/2. In either case, if a punch buffer parity check or a hole count check occurs, the card is stacked in the NP pocket.

### Special Write Conditions — 1402

1. The buffer contents are parity checked during punching and, if an error is detected, the buffer check indicator in the 1414 is turned on. On the next select punch instruction, the redundancy check indicator in the channel is turned on. If a punch buffer parity occurs, the card is placed in the NP pocket.

2. As the buffer is read out, a hole count is retained by the 1414. On the next card feed cycle, the card passes the punch check station and the holes are again counted and compared with the previous hole count. If the comparison is not equal, the hole count check indicator is turned on. If a select punch instruction is given and the hole count check is on, the redundancy check indicator in the CPU is turned on and the card is placed in the NP pocket.

3. Character parity is checked against word parity as the data are placed in the punch buffer. If an error is detected, the word parity indicator in the channel is turned on.

4. When a select instruction is given to the punch, the not-ready and busy conditions are the same as with the reader. If there has been a hole count or a parity error on the previous card feed cycle, the redundancy check indicator in the CPU is turned on and the card is placed in the NP pocket.

### Instructions

Normally, two instructions, read select (RDS) and write select (WRS), are used to select the device for reading or writing. When it is advantageous to have program compatibility with IBM 7090/7094 systems, two different instructions may be used: prepare to read (PRD) and prepare to write (PWR). Both of these instructions, when executed on 7040/7044 systems are identical in all respects to the read and write select instructions. When either the PRD or PWR instructions are executed on 7090/7094 systems, a store and trap operation results, providing a convenient linkage to a subroutine. This subroutine may then simulate input/output functions on the 7090/7094 systems. The RDS and WRS formats are described with magnetic tape operation.

In the following instruction descriptions, Y designates the address part of the instruction. This field selects the data channel and device to be used on that

channel. If the device is attached to channel A, additional information contained in positions 15-17(V) is needed to specify the proper channel A adapter; positions 15-17 are not interpreted by 7904 Data Channels.

Since both binary or BCD coded information is possible, the address part of the select instruction not only designates the channel, channel adapter, and device attached to that channel, but also designates whether binary or BCD coded information is to be processed. With a BCD address, the information is automatically translated to the internal binary coding scheme of the computer. With a binary address, no translation occurs. Therefore, a binary address may be used to process any type of card coding, but the program will then have to do whatever translation is necessary so that the computer can operate on the information and reach a meaningful result.

### Prepare to Read — PRD Y,T,V

This instruction prepares the channel to read information from the input device specified by Y and V into core storage. Only positions 28-35 of the address part of the PRD are subject to address modification. Bit 14 of the PRD must be a 0 bit.

The Y part (positions 21-35) of the PRD selects the mode of reading and is the same for the 1622 and the 1402: 01210 for BCD and 01230 for binary. The V part (15-17) of the PRD selects which card reader is to be used.

### Prepare to Write — PWR Y,T,V

This instruction causes the channel to prepare to write information from core storage to the output device specified by Y and V. Position 13 of the V field is used by the 1402 card punch and bit position 14 of the PWR must contain a 0 bit. Only positions 28-35 of the address part are subject to address modification.

The Y part of the PWR selects the mode of recording and is the same for the 1622 and the 1402: 01210 for BCD and 01230 for binary. The V part selects which card punch is to be used. Position 13 of a PWR is only used with the 1402 and selects which punch stacker pocket is to be used: a 1 for pocket 8/2 and a 0 for pocket 4.

The V part is shown below as a five-position number (binary) and designates the 1402 cards being selected for the 8/2 pocket:

13 14 15 16 17 — V part of PWR  
1 0 0 1 1 — Code for 8/2 pocket

This binary number, converted to a decimal number, is then placed in the V part of the instruction as:

PWR 1230,,19

An example of the PRD instruction (as used with the 1622) is shown in Figure 132. Assume that an IBM

coded card with 80 characters is to be read. As previously described, 13 full word locations and 12 positions of a 14th location are used to hold 80 BCD characters.

To record a card in column binary on the 1402 and put the punched cards in the 8/2 select stacker pocket, the routine shown in Figure 133 could be used.

### Transfer on Channel A Device in Operation — TDOA Y,T,V

This instruction tests the busy status of individual devices specified by the V part of the TDOA. The information contained in the V part (positions 15-17) shows:

V	DEVICE TESTED
1	Reader (1622 or 1402)
2	Punch (1622 or 1402)
3	Printer (1403)
4	Console Typewriter
5	On-line 1401 System

When the TDOA is executed, the V part is sampled. The adapter associated with the device is selected and the

proper busy indicator is tested. If the device is in operation, the computer takes its next instruction from Y. If the device is not in operation, the computer takes the next sequential instruction. The TDOA will always transfer if channel A is in use, regardless of the status of the device being tested.

### Select Instructions

Two select instructions, sense and control, are similar in operation to the read and write instructions except that they do not result in movement of input/output data. If either of these instructions is executed on an IBM 7090 or 7094 System, a store and trap operation occurs as with the PRD and PWR instructions.

### Sense Select — SEN Y,T,V

This instruction causes the channel to prepare to read status data into core storage from the device specified

Location		Operation	Address, Tag, Decrement/Count	Comments	Identification		
1	2	6	7	8	72	73	80
		PRD	648,,3	Selects the 1622 for BCD read.			
		RCHA	ICOM	Gets the ICOM command.			
ICOM		BCD	INPUT,,14	Moves 14 words into locations INPUT, INPUT+1, INPUT+2, etc.			

Figure 132. 1622 BCD Read Routine

Location		Operation	Address, Tag, Decrement/Count	Comments	Identification		
1	2	6	7	8	72	73	80
		PWR	665,,19	Selects the 1402 for writing and the 8/2 pocket.			
		RCHA	OCOM	Gets the OCOM command.			
OCOM		BCD	OUT,,27	Moves 27 words from storage locations OUT, OUT+1, OUT+2, etc.			

Figure 133. 1402 Binary Write Routine

Location		Operation	Address, Tag, Decrement/Count	Comments	Identification		
1	2	6	7	8	72	73	80
		SEN	658	Status data are needed from tape 2 on channel A.			
		RCHA	IOCOM	Channel command.			
IOCOM		DEC	000001005000	Places one word of status data in core location 5000 (positions S, 1-5).			

Figure 134. Status Data Example

by Y and V. Status data are broadly defined as information about the status of the device being addressed by the SEN.

In the V field, position 13 is used to designate an input/output buffer when a 1414-3, -4, or -5 Input/Output Synchronizer is addressed. A 0 bit selects input buffers; a 1 bit selects output buffers. Position 14 must contain a 1 bit and only positions 28-35 of the SEN are subject to address modification. If this instruction addresses a device using the BCD mode address, no code translation occurs between the device and the computer.

The instruction string shown in Figure 134 shows tape unit 2, attached to data channel A, addressed by a SEN instruction. Status data from the tape unit are to be placed in core location 5000.

### Ready Test

When a SEN addresses a device and is followed by a RCH that loads an IORD with a word count greater than zero, the following status data are automatically stored in the S, 1-5 positions of the addressed storage location (start address) of the IORD. Figure 135 shows the status data in binary format.

### Control Select – CTR Y,T,V

This instruction causes the channel to prepare to send control data to the device specified by Y and V of the CTR. Position 14 of the V field must be a 1 bit, and only positions 28-35 of the CTR are subject to address modification. Use of the CTR instruction is shown in the 1301 Disk Storage section.

Device	S12345	Comment
Magnetic Tape on all	010000 000100 000001	Unit is not ready Unit is rewinding Unit is at load point
Channel A Devices 1622 1414-3 or -4	010000 010000 000100 000010 000001	Unit is busy or not ready Unit is not ready Unit is busy Condition (EOR, EOF, etc.) No translation (column binary format for 1402) Not ready or busy
1401 on line	010000	
7904 Data Channels I/O Channel Adapter	010000	Not operational. All normal status data are stored if device and adapter are operational

Figure 135. Status Data

As an example in use of the SEN instruction, assume that tape unit 3 attached to data channel B is to be tested to find out if it is rewinding. Figure 136 shows an instruction string that would accomplish this.

### IBM 1403 Printer

This unit can produce output documents at 600 printed lines per minute. A dual-speed, paper tape and computer controlled carriage permits high-speed skipping of the printer paper at 75 inches per second for skips of more than eight printed lines. As with the 1402 Card Read Punch, the 1403 printer has its own buffer storage within the 1414-3 or -4 Input/Output Synchronizer.

### Print Operation

The IBM 1403 Printer has 100 printing positions per line, with an additional 32 positions per line available as an optional feature. Transfer of print characters is under control of the IORD channel command. If the

Location	Operation	Address, Tag, Decrement/Count	Comments	Identification
1 2 4 7 8				72 73 80
TEST	SEN	1171	Select tape 3 on channel B.	
	RCHB	10COM	Load command to put the status data into core storage.	
	CLA	REWND	Status data image for rewind condition.	
	CCS	10STO,,0	Where status data were placed in storage.	
	TRA	CONT	Continue program, not rewinding.	
	TRA	TEST	To test or other procedure in case of	
	Continue		rewind condition.	
10COM	BCD	10STO,,1		
10STO	BSS	1		
REWND	BCI	1,bbbbbbM	(b is the blank character)	

Figure 136. SEN Program Example

word count is greater than 16 or 22, only the first 100 or 132 characters, respectively, are transferred to the print buffer. If the word count is equal to or less than 16 or 22, the print buffer is filled out with blanks. When the buffer is full, the channel signals to print the line. The channel is then disconnected and the CPU proceeds to the next sequential instruction. For a normal print operation, a WRS is used, followed by a RCH, which loads the command into the channel registers. The rest of the operation is similar to the punch operation.

### Carriage Control Operation

The printer carriage is controlled by a special control character. This character is sent to the printer by a control (CTR) instruction followed by a RCHA, which loads the IOBD command with a word count of one (a word count greater than one is treated as one). The character defined in bit positions S, 1-5 of the data word is used. The channel then disconnects and the CPU executes the next sequential instruction. Figure 137 shows the control characters.

### Special Print Conditions

1. Buffer contents are checked for parity during printing. If a parity error is detected, the buffer parity check indicator in the I414 is turned on. On the next select printer operation, the redundancy check indicator in the CPU is turned on.
2. During printing, the printer circuitry determines if the proper character has been printed. If an error is sensed, the next printer select instruction turns on the redundancy check indicator in the CPU.
3. The printer-not-ready condition is caused by printer out of forms, printer in manual status (off-line),

or printer power off. When the printer is selected by the CPU and is not ready, the CPU halts operation.

4. The printer-busy condition occurs when the printer is selected and the previous line is still being printed. If the printer is selected while in busy status, the CPU waits for a not-busy condition.

5. If a parity or print error has been detected by the I414 during the preceding print cycle, the channel redundancy check indicator in the CPU is turned on.

6. Character parity is checked against word parity as the data are placed in the print buffer. If an error is detected, the word parity indicator in the channel is turned on.

### Console Typewriter

The console typewriter, which is available for output operations only, has a maximum rate of 15 upper-case characters per second. Information is printed serially by character under program control. A blank character results in a space function on the typewriter. Automatic carriage return is provided at the end of every line and at the end of a record.

### Typewriter Busy

The typewriter busy status, which may be tested by TDOA instruction, will be present during carriage returns and, in single-character operation, for about 20 milliseconds after channel-A-in-use indicator is turned off. When the typewriter is selected with a CTR instruction, the instruction only waits if channel A is in use. If channel A is not in use, the CTR will be executed even if the typewriter is busy. This will result in leaving the channel-A-in-use indicator on while the typewriter is busy during the first part of the next print cycle.

### Write Operation

A prepare-to-write instruction (PWR), addressing channel A and the typewriter, selects the typewriter for use. The channel initiates data transfer by sending the first byte on the write bus. The typewriter recognizes the presence of data on the write bus and takes a print cycle. When the print cycle is completed, a service request is generated for the next byte; all succeeding characters are transmitted in like manner. The write operation is completed when the word count goes to zero. This initiates a carriage return.

### Single Character Operation

The typewriter may be made to type a single character and not carriage return. This is accomplished by a control select instruction (CTR) selecting the typewriter

Control Character	Function: Immediate Skip to	Control Character	Function: Skip After Print to
1	Channel 1	A	Channel 1
2	Channel 2	B	Channel 2
3	Channel 3	C	Channel 3
4	Channel 4	D	Channel 4
5	Channel 5	E	Channel 5
6	Channel 6	F	Channel 6
7	Channel 7	G	Channel 7
8	Channel 8	H	Channel 8
9	Channel 9	I	Channel 9
0	Channel 10	?	Channel 10
#	Channel 11	.	Channel 11
@	Channel 12	π	Channel 12
	Immediate Space		After Print Space
J	0 Space	/	0 Space
K	1 Spaces	S	1 Spaces
L	2 Spaces	T	2 Spaces

Figure 137. BCD Carriage Control Characters

followed by a RCHA, which loads a command. When the command has fetched the data word, bits S, 1-5 will be loaded in the typewriter buffer and the RCHA will end operation. Channel A will remain in use, however, until the character has been typed. The next CTR and RCHA must be given within 28 milliseconds after the channel-in-use indicator is turned off to maintain full typewriter speed. A carriage return will not occur unless the end of a line has been reached. If the command has a word count of zero, no character will be typed and the typewriter will carriage return.

**Shifting**

To print all 64 characters, it is necessary to translate certain characters to upper case. All upper-case characters have either 0 bits in the 8, 4, 2, and 1 positions or have 1 bits in the 8 and 4 positions. During shifting from upper to lower case or vice versa, an additional

character print time is required to accomplish the shift. Blank, which is interpreted as a space, does not require shifting, no matter if the typewriter is in upper or lower case. Figure 138 shows upper-case characters.

Report	Programming	Report	Programming
⌘	⌘	@	'
:	:	#	#
>	>	□	)
√	√	┌	┌
<	<	&	+
≠	≠	/	/
-	-	%	]
*	*		
;	;		
Δ	Δ		

Figure 138. Typewriter Upper-Case Characters

## Disk Storage and Other Optional Features

### **IBM 1301 and IBM 7631**

The 1301 Disk Storage and the 7631 File Control are available for the IBM 7040 and 7044 Data Processing Systems, IBM 1410 Data Processing System, and other IBM 7000 Series Systems.

The 1301 Disk Storage is available in two models:

- Model 1 Single module, providing capacity for 27,960,000 characters
- Model 2 Two modules, providing capacity for 55,920,000 characters

Other operating characteristics are:

Positioning of Access Mechanism	50-180 milliseconds
Average Rotational Delay	17 milliseconds
Characters for One Access Positioning	111,840 maximum
Instantaneous Character Rate	90,100 per second
Characters per Track	2,796 maximum

The 7631 File Control is available in four models:

- Model 1 For use with an IBM 1401 system
- Model 2 For use with 7000 series systems
- Model 3 For shared use between a 7000 and 1410 system
- Model 4 For shared use between two 7000 series systems

### **Magnetic Disk Recording**

The magnetic disk is a thin metal disk coated on both sides with magnetic recording material. The 25 disks are mounted on a vertical shaft and are slightly separated from each other to provide space for the movement of read/write assemblies between them. The shaft revolves, spinning the disks at a maximum of 1,790 revolutions per minute.

Data are stored as magnetized spots in concentric tracks on both upper and lower surfaces of each disk. There are 250 tracks on each surface for storing data. The tracks are accessible for reading and writing by the read/write heads, which move horizontally between the spinning disks.

The read/write heads are mounted on an access mechanism, which has 24 arms arranged like teeth on a comb. The arms move horizontally between the disks (no vertical motion is involved). Two read/write heads are on each arm. One head serves the bottom surface of the upper disk; the other head serves the top surface of the next lower disk. Thus, it is possible to read or write on either side of a disk.

The magnetic disk data surface can be used many times. Each time new data are stored on a track, the old data are erased as the new are recorded. The recorded data may be read as often as desired; data remain recorded in the tracks of a disk until new data are written over the old.

Although the total number of character positions of each track is fixed, the arrangement as to the number of records and the number of characters per record can be varied to suit the needs of the application. Thus, data can be stored on a track in any convenient arrangement within the limitations of track requirements. Addresses must be provided to identify the track and the individual records to the computer; also, space must be provided in the form of gaps to separate address and record areas.

The format track provides a means of defining and monitoring the address, record, and gap areas for the data tracks. In the 1301, one format track monitors 40 associated data tracks. The format track can be written and rewritten to describe the desired data track format as often as required to suit the needs of the user.

A disk storage module is composed of the stack of 25 magnetic disks and its associated access mechanism. Of the 25 disks, 20 disks (40 surfaces) are used to store data; of the other ten surfaces, six are alternate surfaces, one is a format surface, one is a clock surface, and the other two are not used. Both the data and format surfaces contain 250 concentric tracks that are accessible for reading and writing.

### **Cylinder Concept**

Since the heads and disk tracks are mechanically aligned one above the other, the vertical alignment of the tracks can be considered as a cylinder of tracks. Thus, with the access mechanism placed in any one of the 250 cylinders, 40 tracks of data (each data surface) are available without further access mechanism motion. For example, for one access mechanism setting, as many as 111,840 characters are available to the computer.

The tracks are numbered sequentially, from the bottom to the top of the cylinder (corresponding to the 40 heads, 00 through 39), starting at the outermost cylinder on a given surface (000) to the innermost cylinder (249). Thus, with large storage areas for reference tables, the data can be conveniently stored in a cylinder of tracks or in a number of adjacent cylinders. This technique reduces access time to a minimum. The cylinder arrangement of tracks also permits the optional feature (cylinder mode) to read or write a cylinder, or part of a cylinder, in one operation.

### Data Track Address

The basic fixed recording area of the 1301 is the data track. The physical make-up of the track limits over-all recording capacity. All data tracks are equal in storage capacity, but the entire recording area cannot be used to store data. A number of track positions are used to identify the track and records to the computer. These positions are called index point, home address, record address, and gaps.

The *Index Point* of the track is the reference point of the track; it indicates both the beginning and end of the track, and synchronizes the disk storage with the computer. The index point of each track is fixed and cannot be altered by the programmer.

The *Home Address* follows the index point and consists of two parts called home address 1 (HA1) and home address 2 (HA2). Home address 1 is a prerecorded four-digit track number (0000-9999); it indicates the physical location of the track within the module and cannot be written by the programmer. Home address 2 is the home address identifier. HA2 is written by the programmer and consists of two or more characters, which can be numeric, alphabetic, or special characters, depending on requirements of the computer system. HA2 can be written to serve any convenient purpose, such as tagging a particular category of records.

The *Record Address* consists of six or more characters, which can be numeric, alphabetic, or special characters; it identifies an individual record on the track. The record address (RA) characters are assigned and written by the programmer to fit any convenient addressing scheme. The home address (used for seek orders) and the record address (used with the prepare to verify single record order) need not be related in any way. Only the numeric portion of the first four record address characters is verified; the next two record address characters are completely verified; additional (more than six) address characters are not verified.

*Gaps*, consisting principally of zero bits, are automatically written between all address and all record areas on a data track to distinguish between addresses and records. The gaps contain check characters and internal synchronization information required for proper operation.

### Format Track

Before the disk can be used for reading or writing, a format track must be written for each cylinder of the disk module. The format track designates how storage space of the tracks of a disk cylinder is to be identified and used. Once established, the format track provides a fixed format for subsequent reading and writing. Data used to create the format track must first be organized in core storage and are then sent to the addressed format track from core storage.

The writing and the layout of format tracks are under control of the programmer. Once written, the format track remains fixed until rewritten. To prevent accidental changes to the format tracks, each disk module has a two-position key-lock switch. A format track can only be written when the switch is in the write position; normally the switch is in the read position.

Each of the 10,000 data tracks must have an index point, one home address, a record address for each record stored on the track, and the necessary gaps to separate the address and the records. Figure 139 shows this layout.

### Operation

The 7631 File Control and 1301 Disk Storage are used to illustrate the use of 7040/7044 instructions as related to an input/output control adapter. The following functions are to be performed.

1. Select the 7631/1301 disk channel.
2. Give the 7631 enough orders to cause one access mechanism to locate itself at some track before telling it to write a record.
3. Write the record.
4. Check the record for validity.

The instruction sequence is:

1. *Control – CTR*: Selects the channel (bits 24-26). A flag bit in position 14 denotes a control instead of a write operation. Address of 02000 or 02020 denotes the control adapter interface.

2. *Reset and Load Channel – RCH*: Brings the location of a control word (the file control order – Seek) from core storage. This order tells what head and tracks are to be used. When the access mechanism has located the proper address, the disk channel causes an attention interrupt. This interrupt must be serviced by a sense instruction.

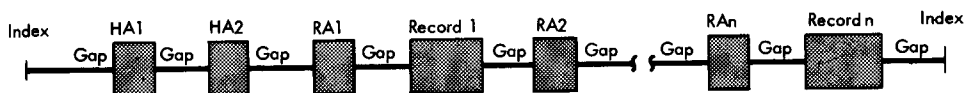


Figure 139. Sample Track Layout

3. *Sense – SEN*: Selects the channel and control adapter interface. Status data, including the attention interrupt, have come from the disk, through the file control, and are now in the data channel.

4. *Reset and Load Channel – RCH*: Brings in the location of another control word. This word takes the status data and places that data in core storage at a specified location. The computer must analyze the status data and determine what action is necessary.

5. *Control – CTR*: Selects the channel and control adapter interface.

6. *Reset and Load Channel – RCH*: Brings the location of a control word (prepare to verify – single record) from core storage. This order verifies the record address when the PWR instruction is given.

7. *Prepare to Write – PWR*: Selects the channel and the control adapter interface. The instruction prepares the file control and the I301 for a write operation.

8. *Reset and Load Channel – RCH*: Brings the location of a control word containing the starting address of the record to be written and a word count for the number of words to be written. Data transmission starts and continues until the word count equals zero. At this point, the record has been written on the I301, but, since validity of the record should be checked, the program must write-check what it has written.

9. *Control – CTR*: Selects the channel and the control adapter interface for the write check operation.

10. *Reset and Load Channel – RCH*: Brings the location of a control word (prepare to write check) from core storage into the file control.

11. *Prepare to Write – PWR*: Selects the channel and control adapter interface for the write check operation.

12. *Reset and Load Channel – RCH*: Brings the location of a control word with the same starting address and word count as was brought into the data

channel with the RCH in step 8. The record is read from the disk and compared bit-for-bit against the record being transmitted again from core storage.

### Programming Examples

Data transmission is accomplished by an RCH of the channel command that transfers the data. For example, if the channel is enabled to interrupt the CPU at completion of a seek operation, the instruction at the “trapped-to” location could contain a transfer to a location that contains:

```
SEN
RCH 1000
```

Location 1000 would contain the channel command with a word count of 2. The routine would then: (1) check the bit pattern of the two words sensed to determine if the operation was a seek and what access arm completed the seek, and (2) transfer control to the proper location according to the condition.

The disk system operates in one of four modes: single record, track record, home address, and cylinder mode (optional feature). Single-record operation is used to read or write a single record with the disk system; a program to read a single record on track 2500 could be like the one in Figure 140.

1. The control instruction (CTR) prepares the channel to receive the file control order in BCD format.

2. The reset and load channel B (RCHB) loads the channel with an IORD with a word count of 2. The address of the IORD contains the file order (seek track 2500) in BCD format.

3. The CPU then continues until the seek is completed, the channel is trapped (not shown) and control is transferred to location 00150.

4. The CPU executes another CTR and a RCHB of an order that contains a prepare to write single record order. This is sent to the file control in BCD format. The

Location		Operation	Address, Tag, Decrement/Count	Comments	Identification		
1	2	6	7	8	72	73	80
00100		CTR	02000	Prepare control unit.			
00101		RCHB	00500	Load seek IORD.			
				This area contains the program to be used while seeking.			
00150		CTR	02000	Prepare to verify.			
00151		RCHB	00503	Record address.			
00152		PRD	02020	Prepare to read.			
00153		RCHB	00506	Load command.			
00500		BCD	501,0,2				
00501		BCI	2,800025000000	Seek track 2500.			
00503		BCD	504,0,2	Prepare to verify.			
00504		BCI	2,8200ANYRD#00	Record address.			
00506		BCD	100,0,700	Read-into location.			

Figure 140. Read Single Record Program Example



file control verifies the address of the order against the record address on the track.

5. The prepare to verify order is followed by a prepare to read and a RCHB that transmits data to core storage. If the address is not verified, the file control signals with an unusual end and no data are transmitted.

In this example, the effect of the trap is not shown. All traps in channel transfer control to a fixed location and store the instruction counter. Certain conditions may be determined by checking the decrement field of this location for a 1 bit in positions:

- 16 For redundancy check
- 14 For word parity
- 12 For an unusual end signal
- 11 For an attention signal

For more detailed information (such as which arm has completed seek) it is necessary to test the file control with a sense instruction followed by a RCH of an IORD with a word count of two. This provides the CPU with control status data.

Track record operation is used to read or write a full track. This mode is also used to write a record address. A routine to read a full track could be the same as the preceding example, except that the file order prepare to verify track is substituted for the prepare to verify record order.

Home address operation transmits or receives all track records, record addresses and home addresses. To operate in this mode, substitute the order: prepare to verify home address.

The other mode is the optional cylinder mode. This mode enables the programmer to read or write all cylinder records with one order, which is substituted for the verify order.

The program in Figure 141: selects the disk system, gives it enough orders to locate the access arm at the desired record, and writes and then verifies the record.

1. Control is passed to the routine from the main program by executing a `TSX` instruction. Another means to pass control is by a `TSL`; this operates similar to a

Location		Operation	Address, Tag, Decrement/Count	Comments	Identification
1	2	4	7	8	72 73 80
	15001	TSX	00200,4	Transfer to 1301 routine.	
	00200	CTR	02000	Prepare control unit.	
	00201	RCHB	00700	Load seek IORD.	
	00202	TRA	1,4	Return to main program (access arm attention occurs here).	
	00203	SEN	02000	Prepare to sense.	
	00204	RCHB	00701	Receive status data.	
	00205	TCOB	00205	Wait for status data.	
	00206	CAL	01002	Is it Attention?	
	00207	ANA	01004	Check attention mask.	
	00210	TZE	03500	No, transfer.	
	00211	CTR	02000	Yes, continue.	
	00212	RCHB	00702	Verify record address.	
	00213	PWR	02020	Prepare to write.	
	00214	RCHB	01007	Write single record.	
	00215	TRA	1,4	Return to main program.	
	00216	CTR	02000	Prepare to write check.	
	00217	RCHB	01010	Load order.	
	00220	PWR	02020	Write check.	
	00221	RCHB	01007		
	00222	TRA	1,4	Return to main program.	
	00700	BCD	1000,0,2		
	00701	BCD	1002,0,2		
	00702	BCD	1005,0,2		
	01000	BCI	2,300020000000	Seek track 2000.	
	01004	OCT	000000001000	Attention mask, module 0.	
	01005	BCI	2,8200ANYRD#00	Verify record address.	
	01007	BCD	10000,0,350	Write output.	
	01010	BCD	01011,0,2	Write check.	
	01011	BCI	2,8600ANYRD#00		

Figure 141. Write and Verify Program Example

tsx but does not involve index registers. (The TSL instruction is described under "Trapping.")

2. The 7631 is selected by the CTR and the channel is loaded with a control command (ORD) whose address contains the file order to seek track 2000.

3. Control is returned to the main program. If a TSL is used, return is accomplished by indirectly addressing the effective address of the TSL.

4. On receipt of an attention signal, control is passed to the file routine. The routine for detecting the attention signal is not shown. The CPU tests the control unit by executing a sense select and, through a channel command, receives two words of sense data. If the trap cause was an attention signal from module 0, a 1 bit is placed in position 3 of the fourth character. The first word of sense data is AND'ed against a mask of all zeros, exclusive of position 3.

5. If it is an attention interrupt, the CPU prepares the control unit for writing by transmitting a prepare to verify single record order. This is immediately followed by a write instruction. If the record address is not verified, the file control signals an unusual end and no data are transferred.

6. Control is again returned to the main program.

7. The channel interrupts the main program at completion of transmission (not illustrated) and control is returned to the file routine.

8. The CPU prepares the file control to write-check by transferring a prepare to write check order. This is followed by a repeat of the write sequence. If the record does not verify, an unusual end results.

9. Control is returned to the main program.

### **Direct Data Connection**

The direct data connection permits connection of non-IBM input/output devices to an IBM 7040 or 7044 Data Processing System through any of the IBM 7904 Data Channels. Transfer of data between such devices and the 7040/7044 is the same as with standard IBM input/output units, with a full word being transferred at a time.

The direct data connection, when installed on the 7040/7044, provides a communication link with analog-digital converters, telegraph or telephone lines, radar, telemeters, microwave links, engine test stands, or display units. The direct data connection consists basically of direct data interrupt, 36 data transfer lines, two parity lines, 20 sense lines, and the necessary control lines. This feature permits real-time or direct transmission of data between core storage, via the 7904, and external devices at data transmission rates up to 62,500 or 133,333 words per second (7040 and 7044, respectively).

A direct data interrupt signal from the input/output device to the computer automatically interrupts normal program execution and transfers program control to storage location 00004.

On interruption, the address of the next normal instruction to be executed replaces the address part of location 00003 so that re-entry to the normal program is possible after processing. The direct data interrupt signal is under control of the enable instruction.

Data transfer between any associated IBM channel input/output device and core storage of the 7040/7044 is accomplished over 36 data lines and one parity line. These lines are brought out to connectors that may be cable-connected to the direct data I/O device.

The sense lines, which are under program control, provide a data transfer between any core storage address and the direct data connection. Ten lines are provided for input control and another ten lines are provided for output control. The sense lines may be used for ten-bit data transfer, multiple I/O units control, coding or decoding units selected, or logic functions.

The direct data connection is installed on any IBM 7904 Data Channel and uses the data register of the data channel as its buffer. The ORD command is used with the direct data connection as in standard 7904 operation.

### **Computer-to-Computer Operation**

The direct data connection may be used for high-speed communication between two 7040/7044 systems. Communication between the computers is started by a present sense lines (PSL) instruction from one computer to the other. Execution of the PSL causes a direct data interrupt at the other computer. The routine that services this direct data interrupt executes a store sense lines (SSL) instruction to determine what information the first computer is sending. The second computer may then respond by executing a PSL instruction, which causes a direct data interrupt in the first computer. By use of the sense lines and direct data interrupts in both computers, the two programs are initialized for communication over the 36-bit direct data interface.

One computer must be placed in read status and the other computer in write status. Once each computer has selected its direct data interface and set up controls for starting address and word count, data transfer automatically occurs between the systems on a demand and response basis without further programming intervention. When the word count in either computer is reduced to zero, the other computer receives an end-of-record signal and both channels disconnect. Word parity errors occurring in one computer set the redundancy check indicator in the other computer, allowing both programs to determine transmission accuracy.

### General Programming Information

The fastest IBM input/output device available as standard equipment on the 7040/7044 System has a data rate of about one word every 66.66 microseconds. If the external device to be used with the direct data connection has a data rate no faster than this figure, no programming restrictions other than the standard rules are applicable. When data rates from these external devices exceed the fastest IBM data rates, other channel activity must be curtailed. To achieve the maximum data rate of 62,500 or 133,333 words per second (7040 and 7044, respectively), all other data channel operation must be stopped.

To determine the maximum data rate possible with a given computer input/output configuration, include one machine cycle for each additional data channel in use plus two machine cycles for the 7040 processing unit or three machine cycles for the 7044 processing unit. Multiply the total number of machine cycles by the cycle time of the computer system (8.0 microseconds for the 7040, 2.5 microseconds for the 7044) to obtain a figure in microseconds. Divide 1,000,000 by this figure to obtain the maximum number of words per second. Allow a safety factor percentage for random fluctuations in computer timings.

### Direct Data Connection Instructions

The attachment of the direct data connection to any of the 7904 Data Channels permits connection of many nonstandard input/output devices or connection of IBM 7040, 7044, 7090, or 7094 Data Processing Systems. Therefore, the direct data connection, when installed on a 7040 or 7044 system, provides a communication link with analog-digital converters, microwave links, engine test stands, or other IBM Data Processing Systems.

Data are transmitted, a full word (36 bits) at a time, from the external device through the direct data connection to core storage. All possible attached devices are termed external devices in this description. The external device has the ability, through the direct data connection, to interrupt normal computer processing when necessary to transfer data to or from core storage. Figure 142 shows data flow for the direct data connection feature.

The external device is selected by an RDS or WRS, with the address specifying a data channel and the direct data connection. Two new instructions are added to the instruction set for setting and testing the 20 sense lines to the external device. These sense lines are, therefore, under program control. Ten lines are for input control and ten for output control. The sense

lines may be used for ten-bit data transfers, multiple external device control, coding or decoding units selected, or for logic functions.

### Present Sense Lines, Channel B — PSLB Y,T

A separate instruction is provided for each 7904 Data Channel and refers to positions 8-17 of the designated storage location Y. The instruction presents this bit configuration in pulse form to the direct data connection. The bit configuration is preceded by an automatic reset pulse on a separate line. Mnemonics for all 7904 Data Channels are: PSLB, PSLC, PSLD, and PSLE.

### Store Sense Lines, Channel B — SSLB Y,T

A separate instruction is provided for each 7904 Data Channel. The instruction samples the static (at rest) sense lines from the direct data external device and stores their information in positions 8-17 of the storage location specified by the Y part of the SSL. A plus voltage level on the lines is decoded as a 1 bit. Mnemonics for all 7904 Data Channels are: SSLB, SSLC, SSLD, and SSLE.

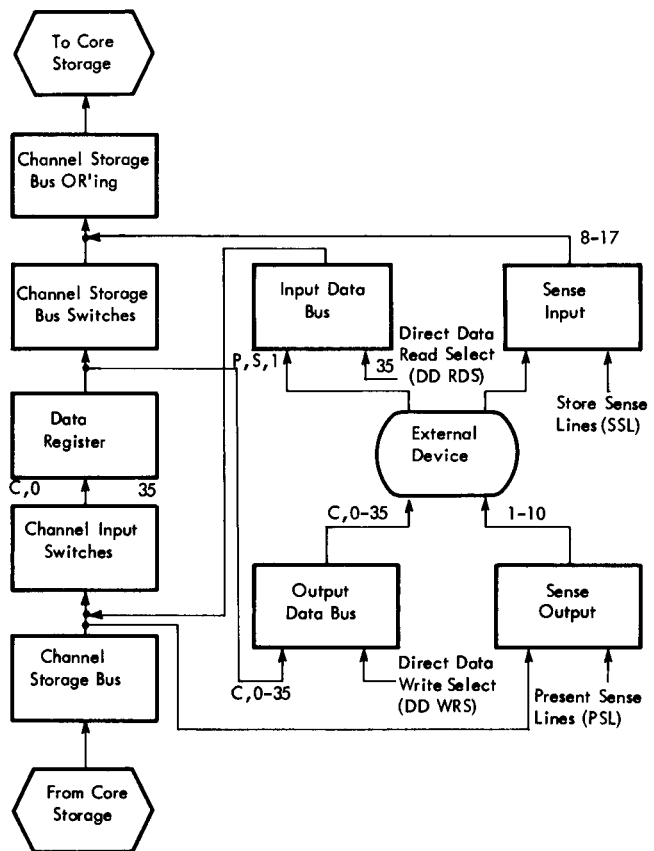


Figure 142. Data Flow, 7904 Data Channel Direct Data Connection

## Storage Protection Instructions

This optional feature provides a flexible means of protecting supervisory or subroutine programs from intrusions by other programs. Two auxiliary registers, set by the supervisory program, are compared against the high-order bits of an effective store address. One register, the count register, determines the number of high-order bits to be examined; the other register, the field register, determines the pattern of bits to be compared against. Violations — attempts to store data in a protected storage area — cause trapping either on an equal or on an unequal compare result, according to the selected protect mode. The format and description of the two protect instructions are:

### Set Protect Mode — SPM Y,T

This instruction places the high-order seven positions of the effective address in the field register and places the C field (positions 32-35) of the SPM in the count register. Bit position 32 sets the mode of protection, and bits 33-35 contain the count of the number of bits to be compared (Figure 143).

If the computer is already in the storage protect mode when the SPM instruction is given, the location of the SPM instruction, plus one, is stored in the address part of core location 0032. Bit 16 of location 0032 is set to a 1 bit (indicating a violation), protect mode is turned off, and the computer takes its next instruction from location 0033.

Indexing may be used to modify the effective address placed in the field register. The count register is not affected by indexing. If this instruction is indirectly addressed, the count register, field register, and tag register are replaced from the indirect location. If a

C Field (Octal)	Bits to be Compared in Each Storage Size				
	32K	16K	8K	4K	
0	None	None	None	None	} Trap if unequal compare result
1	21	None	None	None	
2	21-22	22	None	None	
3	21-23	22-23	23	None	
4	21-24	22-24	23-24	24	
5	21-25	22-25	23-25	24-25	
6	21-26	22-26	23-26	24-26	
7	21-27	22-27	23-27	24-27	
10	None	None	None	None	} Trap if equal compare result
11	21	None	None	None	
12	21-22	22	None	None	
13	21-23	22-23	23	None	
14	21-24	22-24	23-24	24	
15	21-25	22-25	23-25	24-25	
16	21-26	22-26	23-26	24-26	
17	21-27	22-27	23-27	24-27	

Note: A comparison of no-bits always results in an equal condition and hence never traps if the unequal mode is selected and always traps on a store operation if the equal mode is selected.

Figure 143. Set Protect Mode Compare Bits

SPM instruction is given on a system that does not have the storage protect feature, a no-operation results and the computer takes the next sequential instruction.

### Release Protect Mode — RPM

The location of the RPM instruction, plus one, replaces positions 21-35 of storage location 0032. Positions S, 1-20 of location 0032 are set to zero. The computer then takes its next instruction from location 0033. If the computer is in storage protect mode, this instruction turns the storage protect mode off and stores a 1 bit in position 15 of location 0032. If the computer is not in protect mode, a 1 bit is stored in position 14 of location 0032.

If this instruction is given on a system that does not have the protect feature, a normal release protect mode trap with memory protect off occurs. (See "Trapping.") Storage protect mode may also be turned off by pressing the reset or clear key on the operator's console. In this case, no trap occurs.

## IBM 1401 Data Processing System

Any IBM 1401 Data Processing System and its input/output devices may be connected to data channel A by using the 1401 special feature Serial Input/Output Adapter (SF 7080). Except for input/output instructions, computer instructions of both systems operate normally.

To start an input/output operation, the 7040 or 7044 must be synchronized with the 1401 program. Synchronization is possible when the 1401 program is in a mode that enables it to respond to a 7040/7044 instruction. The 1401 informs the 7040/7044 that it is in this mode by executing the KE instruction. This instruction sets an indicator (1401 in loop) in channel A. When the 7040/7044 executes a TDOA instruction for the 1401, the indicator status determines if the program transfers (if the indicator is off, the program transfers).

When any select instruction (RDS, WRS, SEN, CTR, BSR, WEF, REW, RUN) is directed to the 1401, that instruction causes the CPU to hang up if the in-loop indicator is off. If the indicator is on, the select instruction turns it off and sends a signal to the 1401. The 1401 program can sense this signal by executing the instruction B(AAA)2. If the signal is present, the 1401 branches to location (AAA); if the signal is not present, the 1401 executes its next sequential instruction. A basic synchronization loop in the 1401 program could be:

LOCATION	INSTRUCTION
X	KE
X + 2	B(AAA)2
X + 7	B(X)

When the instruction at  $X + 2$  branches, the 1401 program should proceed to a routine that selects its serial I/O adapter to read six bytes. When the serial I/O adapter is selected, the 7040/7044 transfers its entire select instruction (without change) to the 1401. The 1401 decodes this instruction to determine the operation and unit involved. If the unit is tape, reader, punch, or printer, the 1401 program sends status data to the 7040/7044. The status data represent conditions encountered while executing the operation; these conditions may be error, end of file, or end of tape.

After transmission of these conditions (or if no conditions exist), the 1401 executes the 1401 instruction `KD` to signal the 7040/7044 to end operation on its current select instruction. If the I/O operation requires data transfer, the 1401 program should proceed to a routine that selects its I/O adapter to read the record.

Data transfer proceeds when the 7040/7044 executes a `RCHA` instruction. The data are transferred at 11.5 microseconds per character and cease when the channel A word count goes to zero or upon an end-of-record signal from the 1401. The 1401 end-of-record signal occurs when the 1401 encounters a group-mark word-mark in its storage. When the data transfer is complete, the 1401 should send status data that represent any condition encountered during the data transfer. The end-operation signal (`KD`) causes the 7040/7044 to end operation on its current `RCHA` instruction.

When the 1401 program wishes to signal the 7040/7044, a `KF` may be given that turns on the 1401 attention trap request in the 7040/7044. If channel A attention is enabled, a channel A trap occurs.

When the 7040/7044 wishes to signal the 1401, a status line is turned on. This line may be tested in the 1401 by using the `B(AAA)3` instruction. The alternate path of the branch instruction can then be used by the 1401 program to interrupt the 7040/7044 by turning on the 1401 ready indicator.

Two different results occur when 7040/7044 select instructions address the 1401:

1. Execution of a `BSR`, `WEF`, `REW`, or `RUN` instruction leaves the channel not busy after the 1401 ends operation with its `KD` instruction.

2. Execution of a `RDS`, `PRD`, `SEN`, `WRS`, `PWR`, `CTR`, or `WBT` instruction leaves the channel in use after the 1401 ends operation with its `KD` instruction.

Two instructions are used to turn the 1401 status line off and on:

#### **Status Line On, Channel A — `SLNA` ,T**

Execution of this instruction turns on the 1401 status line. The line may be tested in the 1401 with a `B(AAA)3` branch instruction. If the line is on, the 1401 program branches. Since the Y part of the `SLNA` is a part of the operation code, modification by indexing may change the operation.

#### **Status Line Off, Channel A — `SLFA` ,T**

Execution of this instruction turns the 1401 status line off. Since the Y part of the `SLFA` is a part of the operation code, modification by indexing may change the operation.

The following 1401 instructions are used to send various conditions to the 7040/7044 system:

<code>KA</code>	Turn on channel A redundancy check indicator.
<code>KB</code>	Turn on channel A end of file indicator.
<code>KC</code>	Turn on channel A end of tape indicator.
<code>KD</code>	End of operation (terminates the 7040/7044 select or <code>RCHA</code> instruction and turns off the 1401 ready indicator).
<code>KE</code>	Turn on channel A 1401 ready indicator.
<code>KF</code>	Turn on channel A 1401 attention trap request.
<code>M%A2BBB R</code>	Select the 7040/7044 and read into storage starting at location <code>BBB</code> (1401 storage).
<code>M%A2BBB W</code>	Select the 7040/7044 and write from 1401 storage starting at location <code>BBB</code> .
<code>B(AAA)2</code>	Branch to location <code>AAA</code> if the 7040/7044 is waiting with a select instruction.
<code>B(AAA)3</code>	Branch to location <code>AAA</code> if the 7040/7044 status line is on.

# Trapping

## Processing Unit Traps

Automatic trapping of the program is used with the 7040 and 7044 systems to signal unusual conditions to the program without requiring special test instructions. With trapping, system status is constantly monitored and, when particular special conditions are detected, normal processing is interrupted and the program is transferred (trapped) to a trap routine.

To identify the causes of trapping and to allow for a return to normal processing, the instruction counter contents are stored at a fixed location in storage, usually with some trap identification data, when a trap is initiated. The program is then transferred to another fixed location.

Core storage locations assigned for trap operations (Figure 144) are, in order of priority:

TYPE OF TRAP	STORE LOCATION	TRAP LOCATION
Interval Timer Reset	00036	00037
Memory Protect Violation	00032	00033
Storage Parity	00040	00041
Instruction Traps:		
Store Location and Trap (STR)	00000	00002
Floating Point (underflow and overflow)	00000	00010
Release Protect Mode (RPM)	00032	00033
Set Protect Mode (SPM – protect mode already on)	00032	00033
Pre-interrupt Memory Protect	00032	00033
Interval Timer Overflow	00006	00007
Direct Data	00003	00004
Channel E	00022	00023
Channel D	00020	00021
Channel C	00016	00017
Channel B	00014	00015
Channel A	00012	00013

## Delayed Traps

Pre-interrupt memory protect, interval timer overflow, direct data, and channel traps are prevented until after execution of the instruction following certain privileged instructions: RDS, PRD, SEN, WRS, PWR, CTR, ENB, RCT, ICT, or SPM. Also, none of the delayed traps can occur between the XEC instruction and the instruction to be executed. A trap can occur after execution of the instruction referred to by the XEC unless the instruction is a privileged instruction.

## Halt and Proceed

If an interval timer reset, pre-interrupt memory protect, interval timer overflow, direct data, or channel trap request occurs after execution of a HPR instruction, the program stop light is turned off and the trap occurs.

The location of the HPR instruction plus one is placed in positions 21-35 of the trap store location.

## Trapping Priority

Interval timer reset, memory protect violation, and storage parity traps do not need to wait until completion of an instruction to cause a trap. Interval timer reset is the highest priority. Memory protect violation and storage parity trap are mutually exclusive in that if the store instruction has a parity error, it is not executed and, if a store is attempted in a protected area, the parity of the location is not checked. The next highest priority are instruction traps, which are all mutually exclusive because the system cannot be executing a floating point instruction and a STR, RPM, or a SPM instruction simultaneously. The same is true of the privileged instructions. SPM is considered a privileged instruction when it does not trap as a violation. Pre-interrupt memory protect trap has priority over interval timer overflow, direct data, and channel traps so that storage protect mode is never on during these trap routines. The data channel farthest from the CPU (cable connection) has the highest priority of the channels. Channel A, being in the CPU, has the lowest priority.

## Interval Timer Reset

Every 16 $\frac{2}{3}$  milliseconds, the interval timer requests two storage cycles to read out location 00005, add one to it, and store the result back in location 00005. These cycles can only occur:

1. Between instructions.
2. During the following instructions, if they have to wait for the channel: RDS, PRD, SEN, WRS, PWR, CTR, BSR, REW, RUN, and WEF.
3. Between unoverlapped cycles of a RCHA.

Undefined instructions and error conditions exist that prevent the interval timer from getting its two storage cycles. If the interval timer makes a second request before getting cycles for the first, an interval timer reset trap occurs.

The computer may halt operation indefinitely in any of the instructions mentioned, or trap inhibit can be left on. In this case, the interval timer still takes its cycles but an interval timer overflow trap cannot occur. When an interval timer overflow trap is requested, the overflow request is used to block more interval timer cycles until after the interval timer overflow trap or an interval timer reset trap occurs. Incrementing of loca-

tion 00005 is not blocked when the computer is in true manual status. The interval timer overflow trap has about 33 milliseconds in which to trap, or an interval timer reset trap occurs.

The interval timer reset trap does not allow completion of the instruction in process. It resets all data channels including channel A. It does not reset the AC or MQ registers. It stores the instruction counter contents (normally the present instruction location plus one) in positions 21-35 of location 00036 and the computer takes its next instruction from location 00037. Trap control is turned off, inhibiting all other traps, and the two waiting interval timer cycle requests are reset. This means that the contents of location 00005 are two less than they should be when an interval timer reset trap occurs. Interval timer reset trap also resets the interval timer overflow trap request if it is on.

### Memory Protect

A memory protect trap occurs if:

1. A RPM instruction is executed (RPM trap).
2. Memory protect mode is on when a SPM instruction is executed (violation trap).
3. The program attempts to store in a protected area while memory protect mode is on and trap inhibit is off (violation trap).
4. Memory protect mode is on and trap inhibit is off and a channel, direct data, or interval timer overflow trap is requested (pre-interrupt memory protect trap).

NOTE: Input operations on any channel are allowed to store anywhere without causing a memory protect trap.

Any of the above traps turn off memory protect mode and store the location of the next instruction in sequence in the address part of location 00032. The computer then takes its next instruction from location 00033. In the case of a pre-interrupt memory protect trap, the delayed trap is executed instead of the instruction located at 00033. The octal number 33 is placed in the address part of the store location appropriate to the trap that caused the pre-interrupt memory protect trap. The following positions of location 00032 are used to identify the cause of the memory protect trap:

- Bit 17 Pre-interrupt memory protect trap.
- Bit 16 Violation trap or SPM executed with protect mode on.
- Bit 15 RPM executed with protect mode on.
- Bit 14 RPM executed with protect mode off.

### Storage Parity

Possible types of core storage cycles are:

*I cycle:* A cycle to read out an instruction.

*IA cycle:* A cycle to read out an indirect address.

*E cycle:* A cycle to read or store in the execution of an instruction.

*B cycle:* A cycle to read out or store information to or from an input/output device on an overlap channel (the store cycle of a SCH and the read-out of an IORD in a RCH are E cycles, not B cycles).

*U cycle:* A cycle to read out or store information to or from an input/output device on channel A (the store cycle of a SCHA and the read-out of the IORD in a RCHA are E cycles, not U cycles).

*C cycle:* An interval timer cycle to either read out or store location 00005 contents.

Since no parity bit is kept within CPU registers, a word that is stored from the CPU has a parity bit generated as it is stored; therefore, CPU cycles are only checked during read cycles. This includes all I and IA cycles and E and C read cycles. If a parity error occurs during a read cycle, the word is placed in storage unchanged. Parity is checked during both read and store operations for B and U cycles. For a parity error on an input/output store cycle, the word is stored with a generated correct parity.

The following partial word store instructions require one I and two E cycles: STA, STL, SAC, SXA, SXD, STD, and TSL. The first E cycle is used to read out and check the location where the store is to take place. If a parity error is detected during this first E cycle, a parity trap occurs and the instruction is not completed. If no error is detected during the first E cycle, the storage word is placed in the SR and the required portion of the SR is replaced with the new information. During the second E cycle, the complete SR is stored and no parity error can occur.

If a parity error occurs during an I or IA cycle, with parity inhibit and trap inhibit off, the instruction is not executed. The location of the instruction in error, plus one, is stored in positions 21-35 of location 00040. The address of the location in error is stored in positions 3-17 of location 00040 and a bit is stored in position 18 to indicate that the error was either an I or IA cycle. The computer then takes its next instruction from location 00041.

If a parity error occurs during an E cycle with parity and trap inhibits off, the instruction is not executed and the location of the error instruction, plus one, is placed in position 21-35 of location 00040. The address of the location in error is placed in positions 3-17 of location 00040, and a bit is placed in position 19 to indicate that the error occurred during an E cycle. The computer takes its next instruction from location 00041.

If a parity error occurs during a C cycle with parity and trap inhibit off, the computer waits until the instruction being executed is completed; then, the location of the next instruction to be executed is placed in positions 21-35 of location 00040. A bit is placed in

position 1 of location 00040 to indicate that the error occurred during a C cycle. Nothing is placed in positions 3-17, because the location in error is 00005 for a C cycle error. The computer then takes its next instruction from location 00041.

If a parity error occurs during an I, IA, E, or C cycle when either parity inhibit or trap inhibit are on, execution of instructions is not interrupted until both parity and trap inhibits are off. At this time, the location of the next instruction to be executed is placed in positions 21-35 of location 00040, and a bit is placed in position S of location 00040 to indicate that a stacked error occurred. The location of the error is not placed in positions 3-17 of location 00040. Bits are placed in positions 1, 18, and 19 to indicate the type of cycle in which the stacked error occurred. More than one of these bits may be stored when multiple errors occur. The computer takes its next instruction from location 00041.

When a parity trap occurs, both parity and trap inhibits are turned on, preventing further traps. If it is desired to enable all traps except parity, a TRT instruction must be executed. To enable parity traps, a TRP instruction is used.

Parity trap occurs only when parity and trap inhibits are off. The positions of location 00040 indicate:

S	A bit in S indicates that an error occurred while trap inhibit and/or parity inhibit were on (stacked)
1	Indicates an interval timer cycle parity error
3-17	Indicates the location in error if the error is not stacked and is not an interval timer cycle error
18	Indicates that an error occurred during an I or IA cycle
19	Indicates that an error occurred during an E cycle
21-35	Indicates the location of the next instruction to be executed for stacked and interval timer errors. Indicates the location, plus one, of the instruction in error for I or IA, or E cycle error (not stacked)

### Release Protect Mode

Execution of the release protect mode (RPM) instruction places the location of the RPM instruction, plus one, in positions 21-35 of location 00032. Positions S, 1-20 are replaced with zeros. The computer then takes its next instruction from location 00033. If the computer is in memory protect mode, this instruction turns the memory protect mode off and stores a one in position 15 of location 00032. If the computer is not in memory protect mode (or the feature is not installed) when this instruction is executed, a one is stored in position 14 of location 00032. Memory protect mode is also turned off by depression of the clear or reset keys.

### Floating Point

During the execution of floating-point instructions, the resultant characteristic in the AC and MQ may exceed eight bit positions (result too large for storage). The capacity is exceeded if the exponent goes beyond

+177<sub>8</sub> or below -200<sub>8</sub>; beyond +177<sub>8</sub> is termed overflow, below -200<sub>8</sub> is underflow.

Overflow and underflow may occur in either the AC or MQ registers. The computer, on sensing underflow or overflow, puts the address, plus one, of the instruction that caused the condition into the address portion of location 00000. A spill indication is stored in the decrement portion of location 00000 as follows:

BIT	MEANING
S	Double precision instruction on system with single precision feature only.
12	Double precision address error
14	Single-precision divide instruction
15	Overflow in AC and/or MQ register
16	AC overflow or underflow
17	MQ overflow or underflow

The computer then takes its next instruction from location 00010.

### Store Location and Trap

Execution of the store location and trap (STR) instruction places the location of the STR instruction, plus one, in positions 21-35 of location 00000. Positions S, 1-20 of location 00000 are replaced with zeros. The computer then takes its next instruction from location 00002.

### Interval Timer Overflow

This feature allows the computer to be interrupted after a predetermined length of time. When the interval timer increments location 00005 and an overflow from position 1 occurs, a trap is requested. This trap cannot occur unless trap control is on. The trap cannot occur between execution of a privileged instruction and the execution of the next instruction. If memory protect mode is on, then a pre-interrupt memory protect trap must occur before the interval timer overflow trap. The contents of the instruction counter (normally the location of the next sequential instruction to be executed in the main program) replace positions 21-35 of location 00006 and the computer takes its next instruction from location 00007.

When an interval timer overflow trap request is waiting, the interval timer is blocked from increasing location 00005 unless the computer is in true manual status. If the interval timer overflow trap request waits more than 33 milliseconds, an interval timer reset trap occurs, which resets the interval timer overflow trap request.

### Direct Data Trap

This feature allows the channels to signal or interrupt processing by trapping the program. When a direct data trap occurs, the contents of the instruction counter (normally the location of the next instruction to be executed) are stored in positions 21-35 of location 00003. Bits indicating which channels are requesting



a direct data trap are stored in the decrement of location 00003. The computer then takes its next instruction from location 00004. The instruction at location 00004 must be an unconditional transfer instruction to be compatible with the IBM 7090 System.

A direct data trap may occur only when trap inhibit is off and channel trap control is on. A direct data trap cannot occur between execution of a privileged instruction and execution of the instruction following the privileged instruction. When memory protect mode is on, the direct data trap cannot occur until after a pre-interrupt memory protect trap. A direct data trap turns off channel trap control and prevents further direct data traps and channel traps from occurring until after the channel trap control is turned back on with an ENB or RCT instruction.

Each channel has a mask register of four bits. One bit controls direct data interrupt requests from that channel. The mask bits can be set to zero or one by the ENB instruction. For each channel, there is also an indicator that can be turned on by the direct data device. A trap occurs if the indicator is on and the direct data mask bit for that channel is a one. When a direct data trap occurs and the indicator is on, a one is stored in its position of the decrement portion of the store location only if its mask bit is a one. When a direct data mask bit is a zero, the direct data indicator associated with it can be turned on, but a one is not stored in the decrement portion of location 00003 if a direct data trap occurs. The channel x direct data indicator is turned off by an RDCX or by the reset, clear, or load keys. Whenever a one is stored, the indicator is turned off. When a trap occurs and the mask bit is a zero, the indicator is not turned off. The following are the bit positions of location 00003 used to reflect indicator status.

Bit 16	Channel B
Bit 15	Channel C
Bit 14	Channel D
Bit 13	Channel E

### Data Channel Traps

This feature allows the data channel to signal or interrupt processing by trapping the computer program. Channel traps may be initiated by:

- Completion of any channel operation
- Redundancy check
- End of file
- Word parity check
- Unusual-end signal from control adapter
- Attention signal from control adapter
- 1401 attention signal (channel A only)
- Tele-Processing equipment interrupt signal
- Unit record equipment interrupt signal (channel A only).

When a channel trap occurs, the contents of the instruction counter are stored in positions 21-35 of the trap store location. Bits indicating the conditions that caused the trap are stored in the decrement portion of the store location. The remainder of the store location is cleared to zeros. The computer then takes its next instruction from the location specified by the instruction counter. This instruction must be an unconditional transfer to be compatible with IBM 7090 programs. The store locations and instruction locations for each channel are:

CHANNEL	STORE LOCATION	INSTRUCTION LOCATION
A	00012	00013
B	00014	00015
C	00016	00017
D	00020	00021
E	00022	00023

A channel trap may occur only when trap inhibit is off and channel trap control is on. A channel trap cannot occur between execution of a privileged instruction and execution of the following instruction. When memory protect mode is on, channel traps cannot occur until after a pre-interrupt memory protect trap. A channel trap turns off channel trap control and inhibits further channel traps and direct data traps until after channel trap control is turned back on with an ENB or RCT instruction.

Each channel has a mask register, which controls conditions that can cause a channel trap. The mask bits can be set to either a one or zero by the ENB instruction. The clear, reset, or load keys set all mask bits to zero. A RDCX sets all four of channel x mask bits to zero.

For each condition that can cause a channel trap there is an indicator that can be turned on and off by certain conditions. A trap occurs if the indicator is on and the mask bit with which it is associated is a one. When a trap occurs, and the indicator is on, a one is stored in its position of the store location. When a mask bit is a zero, the indicator associated with it can be turned on and off but a one is not stored when a trap occurs. All indicators in channel x are turned off by execution of a RDCX. They are also turned off by the reset, clear, or load keys.

Whenever a one is stored, the indicator is turned off. When a trap occurs and the mask bit is a zero, the indicator is not turned off.

### Channel Trap Stores

When a channel trap occurs, the following bits may be stored in the decrement of the store location. With each bit is a description of the indicator associated with the trap. More than one indicator may signal a trap and store its bit at the same time; therefore, all bit

positions should be scanned rather than stopping at the first position that is found to be a one. The bit positions and indicator names are:

BIT POSITION IN THE STORE LOCATION	INDICATOR NAME	MASK BIT NAME
17	Operation Complete	Operation
16	Redundancy Check	Parity
15	End of File	Operation
14	Word Parity	Parity or Operation
12	Unusual End	Operation
11	I/O Adapter Attention	Attention
10	1401 Attention	Attention
9	Tele-Processing Interrupt	Attention
8	Unit Record Interrupt	Unit Record

*Bit 17, Operation Complete* is turned on whenever the channel-in-use indicator is turned from on to off. This occurs at completion of every read, write, sense, and control operation (end of data transfer), or when the magnetic tape unit has completed a BSR, WBT, or WEF, or started a REW or RUN. This indicator is turned off whenever the channel-in-use indicator is turned on. This indicator is masked by the operation mask bit.

*Bit 16, Redundancy Check* is turned on by a parity check received from the I/O device, or by byte parity check in the channel. When the channel x parity mask bit is a zero, this indicator may be tested and turned off by a TRCX. When the channel x parity mask bit is set to one, a TRCX does not transfer and does not turn off this indicator. Whenever the parity mask bit is a one and the redundancy check indicator is on, the channel stops the transfer of data to or from storage. The channel address register contains the address, plus one, of the last word transferred. A trap or store operation does not occur unless the channel is not in use. For read operations, the channel remains in use for the entire record, even though data are not transferred to storage. This indicator is masked by the parity mask bit.

*Bit 15, End of File* is turned on by the end-of-file signal from the I/O device. When the channel x operation mask bit is a zero, this indicator may be tested and turned off by a TEFX. When the operation mask bit is a one, the TEFX does not transfer and does not turn the indicator off. A trap or store operation does not occur unless the channel is not in use. This indicator is masked by the operation mask bit.

*Bit 14, Word Parity* is turned on by a word parity error during read or write (U or B) cycles to storage. It may also be turned on during channel write operations by checking the 37th bit of a word with the sum of the six parity bits of a disassembled word. Whenever the parity mask bit is a one and the word parity indicator is on, the channel stops data transfer to or from storage. The channel address register contains the address, plus one, of the last word transferred. Therefore, if the parity enable bit is a one when an invalid

word is fetched from storage on a channel write operation, an SCHX stores one beyond the address of the invalid word. A trap or store operation does not occur unless the channel is not in use. When the channel is not in use, if either the parity mask bit or the operation mask bit is a one, the indicator may signal a trap and store. (NOTE: This bit may be stored under control of two different mask bits. The difference between the two is that the parity mask bit allows the channel to stop transmission when an error occurs.)

*Bit 12, Unusual End (Tape Word Incomplete)* is turned on at the end of a tape read or write operation if the total number of characters was not a multiple of six. A tape record that is not a multiple of six characters usually represents a tape read error. This indicator is not set when an end of file is read. If this condition occurs while writing tape, a malfunction is indicated. This indicator is also turned on by the I/O adapter unusual end signal at completion of an IORD to indicate an unusual condition. A sense operation is usually required to determine the condition. This indicator is masked by the operation mask bit and cannot signal a trap unless the channel is not in use.

*Bit 11, Attention* is turned on by the I/O adapter attention signal. This indicator is masked by the attention mask bit and can signal a trap and store even when the channel is in use. This indicator is used with channels B through E.

*Bit 10, 1401 Attention* is turned on by the 1401 and masked by the attention mask bit. This indicator can signal a trap even when channel A is in use, but execution of a RCH cannot be interrupted. The indicator is used with channel A only.

*Bit 9, Tele-Processing Interrupt* is turned on whenever an inquiry buffer in the 1414-4 or 1414-5 has a message waiting, or when an output buffer has emptied. Included in this area are local inquiry, telegraph type units, and IBM 1009 Data Transmission Unit. This indicator is masked by the attention mask bit and can signal a trap even if the channel is in use. The indicator is used with channel A only.

*Bit 8, Unit Record Interrupt* is turned on whenever the following devices on the 1414-3 or 4 have completed their cycle: card reader buffer full, paper tape reader full, card punch buffer empty, or printer buffer empty. This indicator is masked by the unit record mask bit and cannot signal a trap unless the channel is not in use. This indicator is used on channel A only.

### Trap Flow Chart

Figure 144 shows logical interaction of various traps and their results, including the conditions that initiate a trap, trap priority, and CPU action. Beginning at the START or "A" box in the upper left corner, it is possible

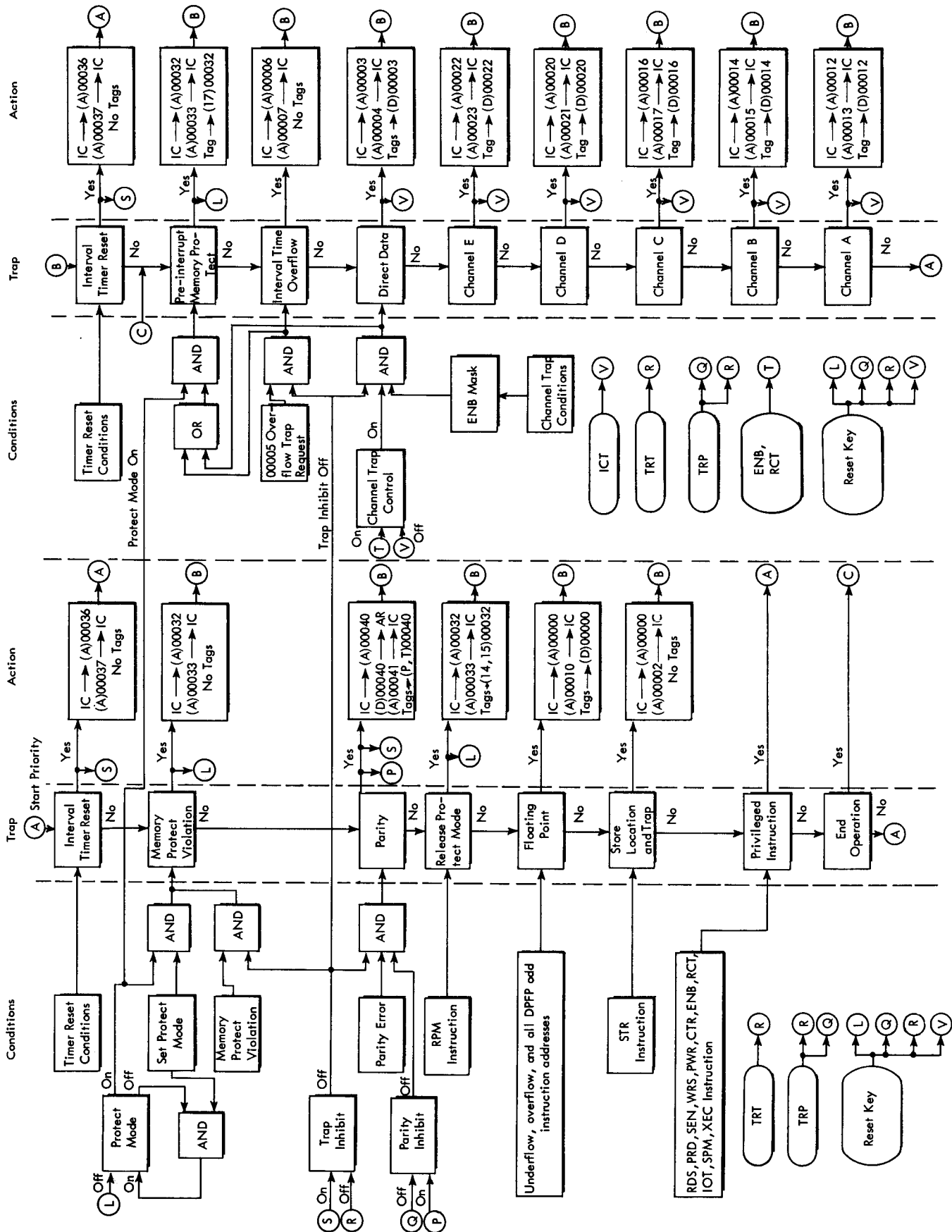


Figure 144. Trap Conditions, Priority, and CPU Action

to trace situations involving multiple trap request, privileged instructions, and so on, and determine the sequence and ultimate action of each condition.

Assume that a parity error occurs during an E cycle. Priority scan circuits pass the parity trap interrupt (from the parity trap box on the left side of Figure 144) out the YES leg. This results in inhibiting further parity trap requests (P output of parity trap box to P input of parity inhibit box), and inhibiting further traps (S output of parity trap box to S input of trap inhibit box). Contents of the instruction counter are placed in the address part of location 00040 (IC→(A)00040), the contents of the address register are placed in the decrement part of location 00040 (AR→(D)00040), and the location of the next instruction (00041) is placed in the instruction counter ((A)00041→IC). After this is accomplished, possible trap requests of the type listed on the right side of Figure 144, are tested (output B of parity trap to input B of interval timer reset). If none of these traps occurs, priority circuits again return to point A (START) and the scanning continues.

### Instructions Used with Trapping

The following instructions are used to store the contents of the location counter or to condition both processing unit and data channel traps.

#### Enable From Y — ENB Y,T

The contents of the storage location specified by Y are used to set the channel mask bits to one or zero. Execution of each ENB cancels the effect of previous ENB instructions. The mask bit name, conditions under which it is enabled, data channel affected, and the mask bit position are shown in Figure 145. The enable instruction turns on channel trap control.

Execution of a trap or inhibit channel traps instruction inhibits all further traps until a new ENB is executed or a restore channel traps instruction is executed. Depression of the reset or clear keys, or execution of a reset data channel instruction, will set all mask bits to zeros.

#### Restore Channel Traps — RCT Y,T

This instruction turns on channel trap control, allowing traps to occur as specified by the previous ENB instruction. It cancels the inhibiting effect of an executed trap or an inhibit channel traps instruction. The address part of the RCT is part of the operation code, and modification by an index register may change the operation itself.

A program using the enable instruction and its mask is shown in Figure 146. Assume that the routine is to

Mask Bit	Conditions Enabled	Channel	Effective if a 1 in Bit Position
Operation	Operation Complete, EOF, Word Parity, Unusual End, or End	A	35
Operation	Operation Complete, EOF, Word Parity, Unusual End, or End	B	34
Operation	Operation Complete, EOF, Word Parity, Unusual End, or End	C	33
Operation	Operation Complete, EOF, Word Parity, Unusual End, or End	D	32
Operation	Operation Complete, EOF, Word Parity, Unusual End, or End	E	31
Direct Data	Direct Data Interrupt	B	25
Direct Data	Direct Data Interrupt	C	24
Direct Data	Direct Data Interrupt	D	23
Direct Data	Direct Data Interrupt	E	22
Parity	Word Parity or Redundancy Check	A	17
Parity	Word Parity or Redundancy Check	B	16
Parity	Word Parity or Redundancy Check	C	15
Parity	Word Parity or Redundancy Check	D	14
Parity	Word Parity or Redundancy Check	E	13
Attention	1401 Interrupt or Tele-Processing Interrupt	A	8
Attention	I/O Adapter	B	7
Attention	I/O Adapter	C	6
Attention	I/O Adapter	D	5
Attention	I/O Adapter	E	4
Unit Record	Unit Record Interrupt	A	5

Figure 145. Enable Instruction Mask Bits

enable for an EOF condition on channel B, word parity or redundancy check condition on channel B, and a 1401 attention signal on channel A.

#### Inhibit Channel Traps — ICT Y,T

This instruction turns off channel trap control, inhibiting all channel traps until a RCT or a new ENB instruction is executed. The Y (address) part of the ICT is a part of the operation code and modification by an index register may change the operation itself.

#### Transfer and Restore Parity and Traps — TRP Y,T

This instruction turns off trap inhibit and parity inhibit and the computer takes its next instruction from Y and proceeds from there.

#### Transfer and Restore Traps — TRT Y,T

The TRT instruction turns off trap inhibit and the computer takes its next instruction from location Y and proceeds from there.

Location		Operation	Address, Tag, Decrement/Count	Comments	Identification		
1	2	4	7	8	72	73	80
		<i>ENB</i>	<i>MASK</i>	This instruction sets up the conditions for enabling according to the mask.			
	<i>MASK</i>	<i>OCT</i>	<i>002002000002</i>	This configuration places a 1-bit in positions 8, 16, and 34 to enable the proper conditions of the problem.			

Figure 146. Enable Routine and Mask Configuration

**Transfer and Store Instruction Location Counter – TSL Y,T**

The location of the TSL instruction, plus one, is stored in positions 21-35 of location Y. Positions S, 1-20 of Y are unchanged. The computer takes its next instruction from location Y + 1 and proceeds from there.

its next instruction from location 00002. The contents of positions 21-35 of the STR instruction are not interpreted by the computer. Note that operation codes of PZE (+0000), MZE (-0000) and ZERO (+0000) are also interpreted and treated as STR instructions.

**Store Location and Trap – STR**

The location of the STR instruction, plus one, replaces positions 21-35 of location 00000. Positions S, 1-20 of location 00000 are set to zero. The computer then takes

**Store Instruction Location Counter – STL Y,T**

The location of the STL instruction, plus one, replaces the contents of Y, positions 21-35. Positions S, 1-20 of Y are unchanged.

# Systems Compatibility

IBM 7040 and 7044 Data Processing Systems are designed so that as the user grows he can, with a minimum of effort, run a large portion of his programs directly on the IBM 7090 or 7094 Data Processing Systems. To assure success, however, the programmer should take certain precautions. There are also certain restrictions to compatibility.

## Compatible Features

1. Data and instruction word formats, addressing, indexing, indirect addressing, accumulator, multiplier-quotient registers, and the instruction counter are compatible.

2. Instructions that are compatible include:

Basic instruction set:

ACL	CLS	LGL	SSP	TMI
ADD	COM	LGR	STA	TNZ
ALS	DCT	LLS	STD	TOV
ANA	DVP	LRS	STL	TPL
ARS	ENK	MPY	STO	TRA
CAL	HPR	ORA	STQ	TZE
CAS	LAS	PBT	STZ	VDP
CHS	LBT	RQL	SUB	VLM
CLA	LDQ	SLW	SWT	XEC

Extended performance set:

AXT	LXD	PDX	SXD	TXH
LAC	PAC	PKA	TIX	TXI
LDC	PAX	PXD	TNX	TXL
LXA	PDC	SXA	TSX	

Single-precision floating-point set:

FAD	FMP	UFA	UFM	UFS
FDP	FSB			

Input/output instructions:

BSR	RCH	RUN	TEF	WEF
ETT	RDS	SCH	TRC	WRS
IOT	REW	TCO		

3. Direct data connection is compatible with a 7090 system equipped with RPQ M90976\*.

4. Traps common to both 7040/7044 and 7090/7094 operate in the same manner: floating point, direct data, storage clock and interval timer and channel traps. Adapter interface attention or unusual-end, unit record, or 1401 traps on channel A may not be available on the 7090 or 7094 systems.

\*Request for price quotation; availability of this feature can be determined only by requesting a price quotation from IBM.

## Incompatible Features

1. The following instructions are not available on the 7090. These instructions have a prefix of -1 (positions S, 1, and 2), which causes a store instruction counter and trap operation on the 7090 or 7094 systems:  
Basic instruction set:

CAP	SLNA	STR	TRT	VMA
SLFA	SLP	TRP	TSL	

Extended performance set:

CCS	MIT	MSP	PLT	TMT
SAC	MSM	PCS		

Memory protect option:

RPM	SPM
-----	-----

Input/output instructions:

CTR	PWR	SEN	ICT	TDOA
PRD				

2. The 7040/7044 double-precision floating-point instructions are not available on the 7090 system. Program modification to include a calling sequence to an interpretive routine is required. These instructions are compatible with the 7094:

DFAD	DFSB	DFMP	DFDP
------	------	------	------

3. Memory parity checking is not available on the 7090 or 7094, but lack of this feature does not affect program execution.

4. Storage protection is not available on the 7090 or 7094. With proper simulation of the RPM and SPM instructions, lack of this feature will not affect normal program execution.

5. Core storage clock and interval timer is similar to RPQ F89349\*, but the interval timer reset trap is not available on the 7090 or 7094.

6. Input/output incompatibility:

7040/7044	7090/7094
Unoverlapped channel A	no
Console typewriter	no
On-line 1401	no
Disk instructions and traps differ	
16 tape units addressable on channel A	10
Channel word parity trap	no

## Detailed Compatibility Information

The following approach to compatibility is not the only one. Note that the 7040 and 7044 are supported by assemblers and compilers that are largely compatible upward to the 7090 and 7094. Most customers, when

converting from 7040/7044 to 7090/7094, would eventually recompile to obtain the most efficient operation. The use of Input/Output Control Systems (iocs) provides an additional interim direct compatibility approach. It is possible to substitute a functionally equivalent 7090/7094 iocs for the 7040/7044 iocs.

### Programs and Routines Other than Input/Output

Programs and routines that do not involve I/O instructions can be operated directly on the 7090 or 7094 if a subroutine is used to simulate the 7040/7044 features not provided on the 7090/7094. Instructions common to both systems operate in the same manner on both systems, and "7040/7044 only" instructions are provided with a prefix of -1 so that they operate as the STR (store location and trap) instruction when operated on the 7090/7094. This provides a linkage to a subroutine for interpreting and simulating these instructions.

#### RESTRICTIONS

1. Programs should be written for 32K core storage for proper indexing operation. This restriction also applies to compatibility between 4K, 8K, 16K, and 32K 7040/7044 systems. The index registers, address register, instruction counter, channel address registers and channel word counters are 15 bits, regardless of the storage size. The storage address register will only contain the significant bits, and high-order unused bits will be ignored when referencing a storage location.

2. There must be sufficient unused core storage to contain the simulation routine. A subset of this routine to handle the 7040/7044 STR (-1000) and zero trap (+0000) is necessary even when operating on the 7040/7044. None of the -1 type instructions should appear in this routine or traps on traps will occur in the 7090/7094.

3. Since the operating speeds of the systems differ, there should be no time dependency. This is true between any of the 7040, 7044, 7090, and 7094 systems.

4. The 7040/7044 memory protection feature cannot be functionally simulated. However, the simulation routine can handle RPM and SPM by setting a pseudo-memory protect word so that these two instructions will behave as they do on the 7040/7044.

5. The 7090 and 7094 do not have word parity trap or interval timer reset traps. Since the TRT and TRP instructions should only occur in these trap routines, the 7090/7094 should never encounter them. These instructions can then be simulated as error halts on the 7090/7094.

6. Any routine that depends upon the interval timer reset trap will not operate on the 7090/7094 even with the RPQ, because this RPQ does not have the reset trap features.

7. The double-precision floating-point arithmetic feature of the 7040/44 is compatible with the 7090/7094 when normalized numbers are used. If un-normalized numbers are used, the results may not be identical.

8. To run a program containing double-precision floating-point on a 7090, it is necessary to reassemble, substituting a calling sequence to an appropriate subroutine for each double-precision instruction.

### Tape-Only Input/Output

Tape-only input/output routines can be operated directly on the 7090/7094 if the following conditions are met:

1. Channel A must be programmed as if it were overlapped (the 7090 must be assured that channel A is no longer in use by means of the TCOA instruction) before data in the I/O area are used.

2. The RCH (reset and load channel) instruction for a given select instruction must issue within the start time of the 7090 tape devices.

### Message Printing

Message printing routines will not operate directly on the printer used with the 7090 and 7094 systems. For this reason, the PWR instruction has a prefix of -1 to cause an STR on the 7090 or 7094. To simplify the 7090/7094 simulation, the RCHA identifying the message should immediately follow the PWR instruction.

### Card Input/Output

Card input/output routines will not operate directly on the card reader and punch used with the 7090 and 7094 systems. It is possible, with off-line equipment or with the 7040/7044 to prepare a unit-record tape corresponding to the card input and to punch cards from a tape prepared by the 7090/7094 in lieu of punching cards. In this case, the 7090/7094 will operate its tape I/O directly by means of the 7040/7044 card I/O instructions. This procedure is facilitated by the assignment of tape addresses to the 7040/7044 card equipment. If the card instructions are to properly operate tape on the 7090/7094, a TCOA instruction must be used as above to interlock the 7090 channel, and the RCHA must issue within the 7090/7094 tape start time. (New error routines may also be desired.)

### On-Line Job Printing

On-line job printing routines may also be handled as tape operations using the 7040/7044 instructions directly, but a format routine is required. The STR

instruction used by the 7040/7044 for this purpose is provided with a prefix of -1 and will be interpreted as an STR when operated on the 7090. The resulting trap may be used as a linkage to a routine for re-aligning the print line for off-line tape-to-printer conversion. Again, the TCOA and RCHA restrictions must be observed.

### Disk Storage Input/Output

Disk storage input/output routines will not operate directly on the 7090 or 7094. For this reason, all disk storage instructions are provided with a prefix of -1 to cause an STR operation in the 7090 or 7094. Through this medium, the disk storage instructions may be converted to 7909 commands to operate disk storage on the 7090 or 7094.

### 1401 Input/Output

If the 1401 I/O routines are a direct replacement for the card and/or tape routines, they are compatible upward as described. Compatibility is relinquished, however, if more than ten tape-addressed units are used or if any on-line editing or other data modification is performed in the 1401.

### 1414-4 Devices

The 1414-4 serial I/O devices are only available on the 7090 or 7094 via the 7909 Data Channel interface adapter channel and 1414-6 interface adapter serial I/O buffer. Although a simulation routine is possible, in general, recompiling is easier and much more efficient. The controlling instructions SEN, PRD, and PWR are provided with a prefix -1 to cause a store and trap on the 7090 or 7094 if desired.

## Programming Compatibility Notes

### PRD, PWR

These two instructions, which are executed identically to RDS and WRS on the 7040 and 7044, should be used when it is desirable to substitute a different I/O routine on the 7090 or 7094 (for example, when using the interface adapter or typewriter). It is recommended that the RCH always be given as the next sequential instruction to simplify the store and trap routine.

Note that on channel A, the interface adapter is used to select the card reader, card punch, printer, or 1401 but, if careful planning is used, each device can be given a different tape address so that the 7090 or 7094, which will ignore the interface adapter, will select a tape. In this case, the RDS and WRS can be used rather

than PRD and PWR, and the program will be directly compatible.

### SEN, CTR

These instructions, which prepare to sense or send control information are used in connection with the interface adapter or with the Tele-Processing equipment on channel A. In general, it is possible to simulate the interface adapter on a 7090 or 7094 with the interface adapter channel. The Tele-Processing equipment on channel A is not usually adaptable to simulation.

### SCHA

An SCHA instruction in the 7040 or 7044 is executed as an SLW instruction. This allows the following sequence of instructions to be executed as a general routine for any channel on either the 7040, 7044, 7090, or 7094.

```

CAL   CHA SAV
RDS
RCHX
SLW   CHA SAV
.
.
TCOX *
.
.
CAL   CHA SAV
SCHX

```

If the RCH was not for channel A, the same thing is stored back in CHA SAV. If it is channel A, then the CHA SAV has the new channel store. This is reloaded with the CAL before the SCH is given, so if it is an SCHA, it will store the AC. Note that the routine also operates correctly on a 7090 or 7094, since the AC is ignored and the TCO (or its equivalent) has assured that the channel is not in use.

### TDOA

There is no way to directly simulate this instruction on the 7090 or 7094, and it is recommended that it be simulated as a TCOA. The programmer should take this into consideration when using this instruction.

### ICT

This instruction can only be simulated as an ENB (zero) instruction on the 7090 or 7094. In addition it must be remembered that several instructions will be executed before the ENB zero is executed. Therefore, if the following were given:

```

ENB   MASK
ICT

```

no channel traps could occur on a 7040 or 7044 between the ENB and ICT instructions, but channel traps



could occur on the 7090 or 7094. Note that if the following is given:

```
ICT
.
.
.
RCT
```

all channel traps would be left disabled after the RCT in the 7090 or 7094. If the following is given:

```
ICT
.
.
.
ENB      MASK
```

the above is corrected but, if a redundancy check occurs on the 7040 or 7044 during the instructions between the ICT and ENB, the channel would disconnect immediately. When run on the 7090 or 7094, it would not disconnect until after the ENB MASK instruction.

### **Trapping Notes**

#### **Trap Stores**

All traps on the 7040 and 7044 use a full word to store information in the trap store location. On the 7090 and 7094, these stores do not affect the prefix or the tag.

#### **Trap Execution**

All traps on the 7040 and 7044 actually transfer to the instruction location. Channel traps and direct data traps on the 7090 and 7094 do not transfer but only execute the instruction at the instruction location. This means that on the 7090 and 7094, if the instruction at the instruction location does not change the contents of the instruction counter, the computer, after executing the instruction, will go back to where it was trapped from. It is necessary, therefore, on the 7040 or 7044 to make sure that this instruction does transfer.

#### **Extra Channel Traps**

Note that any operation that causes a channel to go in use will request a channel trap on the 7040 or 7044. On the 7090 or 7094, only an IORT or IOST will cause a trap at their completion, and then only when no LCH instruction is waiting. These extra traps may be used to advantage on the 7040 and 7044 without forsaking compatibility. To do this, it is necessary to never depend on a channel trap. For instance, do not use a transfer to itself waiting for a trap. Instead, when the program finds itself waiting for the completion of an operation, it is necessary to enter the normal trap routine and take over. This will result in slower operation on the 7090 and 7094.

# Programming Examples

The programming examples included in this section are those thought to be of interest to an experienced programmer. No particular sequence is maintained and the range of problems extends from basic to advanced coding techniques. Both machine symbolic and programming system languages are used. In some examples, flow charts, actual print-outs, and look-up tables are used.

### Example 1

*Given:* Ten ai's and two b's (b1 and b2); where  $b2 > b1$  and  $b1 > 0$ . Assume  $a + 0 > -0$ .

*Problem:* How many of the ai's satisfy the following?

- CASE 1  $0 \leq a_i < b_1$
- CASE 2  $b_1 \leq a_i < b_2$
- CASE 3  $a_i \geq b_2$

Store results in CASE1, CASE2 and CASE3 storage locations.

*Input:* Punched card recorded in IBM card code – 10 ai's followed by the two b's.

*Output:* Typewriter – Contents of CASE1, CASE2, and CASE3.

Figure 147 is a flow chart for program example 1.

The following print-out was received using the 7040/7044 Basic 4K assembly program:

Machine Loc	Octal Oper	Addr	Symbolic Loc	Oper	Addr
00454	060000	000517	START	ORC	454
00455	060000	000520		STZ	CASE1
00456	060000	000521		STZ	CASE2
00457	076203	001210		RDS	CR,,3
00460	054000	000515		RCHA	ICOM
00461	077400	100012		AXT	10,1
00462	050000	100536	LOOP	CLA	A + 10,1
00463	034000	000537		CAS	B2
00464	002000	000500		TRA	C3
00465	002000	000500		TRA	C3
00466	034000	000536		CAS	B1
00467	002000	000504		TRA	C2
00470	002000	000504		TRA	C2
00471	034000	000522		CAS	ZERO
00472	002000	000510		TRA	C1
00473	002000	000510		TRA	C1
00474	200001	100462	TEST	TIX	LOOP,1,1
00475	-176604	001000		PWR	512,,4
00476	054000	000516		RCHA	OUT
00477	042000	000000		HPR	
00500	050000	000517	C1	CLA	CASE1
00501	040000	000523		ADD	ONE
00502	060100	000517		STO	CASE1
00503	002000	000474		TRA	TEST
00504	050000	000520	C2	CLA	CASE2
00505	040000	000523		ADD	ONE
00506	060100	000520		STO	CASE2
00507	002000	000474		TRA	TEST
00510	050000	000521	C3	CLA	CASE3
00511	040000	000523		ADD	ONE
00512	060100	000521		STO	CASE3
00513	002000	000474		TRA	TEST
00514	002000	000454		TRA	START

Machine Loc	Octal Oper	Addr	Symbolic Loc	Oper	Addr
00515	300020	000524	ICOM	IORD	A,,16
00516	300003	000517	OUT	IORD	CASE1,,3
			CR	BOOL	1210
005170000			CASE1	BSS	1
005200000			CASE2	BSS	1
005210000			CASE3	BSS	1
00522	000000	000000	ZERO	DEC	0
00523	000000	000001	ONE	DEC	1
005240000			A	BSS	10
005360000			B1	BSS	1
005370000			B2	BSS	1
004540000			END		START

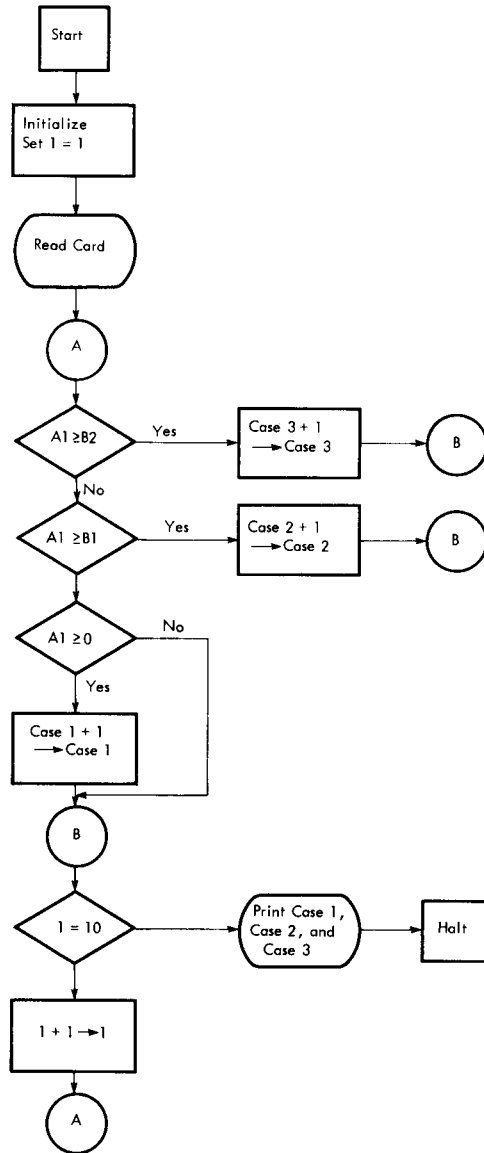


Figure 147. Flow Chart for Example 1

### Example 2

*Problem:* Convert a binary word to six BCD characters. The program could be coded:

Location				Operation	Address, Tag, Decrement/Count	Comments	Identification
1	2	4	7	8			72 73 80
				CLA	BINWD	Put the binary word in accumulator.	
				L D Q	ZERO	Load the MQ with zeros.	
				V D P	TABLE+5,,4	Variable divide using table scaling	
				V D P	TABLE+4,,6	factor. Each division converts six	
				V D P	TABLE+3,,6	binary bits to one BCD digit. The	
				V D P	TABLE+2,,6	count (V) is shown as the last digit	
				V D P	TABLE+1,,6	in the address of the instructions.	
				V D P	TABLE,,6	Sixth BCD digit is converted.	
				S T Q	BCDWD	Store the BCD word.	
				H P R		Stop.	
	TABLE			O C T	200000000000	Scaling Factors:	
				O C T	024000000000		
				O C T	003100000000		
				O C T	000372000000		
				O C T	000047040000		
				O C T	000006065000		
	ZERO			O C T	000000000000		

### Example 3

*Problem:* Convert six BCD digits to a binary word. The program could be coded:

Location				Operation	Address, Tag, Decrement/Count	Comments	Identification
1	2	4	7	8			72 73 80
				L D Q	BCDWD	Put BCD word into the MQ.	
				V L M	TABLE,,6	Multiply using the proper scaling	
				V M A	TABLE+1,,6	factor from the table. Each	
				V M A	TABLE+2,,6	multiply converts one BCD digit	
				V M A	TABLE+3,,6	to six binary bits. The count (V) is	
				V M A	TABLE+4,,6	shown as the last number in the	
				V M A	TABLE+5,,4	instruction's address part.	
				S T Q	BINWD	Store the converted binary word.	
				H P R		Stop.	
	TABLE			O C T	200000000000	Scaling Factors:	
				O C T	024000000000		
				O C T	003100000000		
				O C T	000372000000		
				O C T	000047040000		
				O C T	000006065000		

### Example 4

*Problem:* To duplicate the information from one magnetic tape on another magnetic tape. Record length is up to 15,000 words. Information recorded on the master tape may be BCD and/or binary. The program assumes that:

1. Master tape is mounted on tape unit B1.
2. Work tape is mounted on tape unit C1.

3. When sense switch 1 is depressed, tapes will rewind and unload.
4. When sense switch 1 is off, the start key must be pressed to copy another file.
5. Program is to be read in from 1402 card reader by using the console entry keys.
6. The console load key must be depressed to start the program.

**Example 4 (Cont'd)**

```

00214 042000 000214 ANEW HPR * HIT START TO COPY ANOTHER FILE
00215 -162307 000541 MSP MMMM15,,7 NEGATS EOF ROUTINE
00216 002000 000227 TRA DUPFIL COPY ANOTHER FILE
00217 077200 002221 BEGIN REW 1169 REWIND B1
00220 006100 000220 TCOB *
00221 077200 003221 REW 1681 REWIND C1
00222 006200 000222 TCOC *
00223 050000 000476 CLA BBBB01
00224 060100 000015 STO 13 CONTAINS AN UNCOND. XFER TO TRAPB ROUTINE
00225 050000 000512 CLA CCCC01
00226 060100 000017 STO 15 CONTAINS AN UNCOND. XFER TO TRAPC ROUTINE
00227 077400 100002 DUPFIL AXT 2,1
00230 076000 002352 RDCB
00231 076000 003352 RDCC
00232 056400 000536 ENB MMMM12 ENABLE ALL TRAPS
00233 050000 100525 CLA MMMM03,1 READ COMMAND WORD
00234 060100 000531 STO MMMM07 SAVE READ COMMAND WORD
00235 052200 000500 XEC BBBB03 READ SELECT B1
00236 -054000 100525 RCHB MMMM03,1 READ B1
00237 -134107 000537 BBUSY PLT MMMM13,,7 IS B BUSY
00240 002000 000237 TRA *-1 YES
00241 -134107 000541 PLT MMMM15,,7 TEST SIGN, IF + SKIP NEXT INSTRUCTION
00242 002000 000466 TRA AFILE XFER TO EOF ROUTINE
00243 -134107 000504 PLT BBBB07,,7 ARE WE IN BCD MODE
00244 002000 000247 TRA BINBIN NO
00245 050000 000515 CLA CCCC04 YES, CHANGE TO BCD MODE
00246 002000 000250 TRA *+2
00247 050000 000514 BINBIN CLA CCCC03
00250 060100 000513 STO CCCC02
00251 -162306 000504 MSM BBBB07,,6 NEGATS BCD MODE
00252 -064000 000533 SCHB MMMM09 STORE CH. B WORD COUNTER AND ADDR. REG.
00253 -053400 200533 LXD MMMM09,2 LOAD INDEX 2 WITH WORD COUNT
00254 -063400 200534 SXD MMMM10,2 STORE INDEX 2
00255 050000 000535 CLA MMMM11 CLEAR AND ADD 15000 IN DEC PORTION OF ACC.
00256 040200 000534 SUB MMMM10 SUBTRACT WORD COUNT
00257 062200 100527 STD MMMM05,1 STORE ABOVE IN DECREMENT OF WRITE COMD. WD.
00260 200001 100262 TIX CDWRD,1,1
00261 077400 100002 AXT 2,1
00262 075400 100000 CDWRD PXA ,1 PLACE CONTENT OF INDEX REG. 1 INTO ACCUM.
00263 076000 000001 LBT 1 TEST BIT 35, IF A 1 SKIP AN INSTRUCTION
00264 002000 000267 TRA *+3 A ZERO IN BIT 35, SKIP 3 INSTRUCTIONS
00265 050000 000525 CLA MMMM03 WRITE COMMAND, CORRELATES WITH MMMM01
00266 002000 000270 TRA WRITE
00267 050000 000526 CLA MMMM04 WRITE COMMAND, CORRELATES WITH MMMM02
00270 060100 100531 WRITE STO MMMM07,1 WRITE COMMAND WORD
00271 060100 000532 STO MMMM08 SAVE WRITE COMMAND WORD
00272 -162306 000540 MSM MMMM14,,6 MAKE CHANNEL C BUSY
00273 052200 000513 XEC CCCC02 WRITE SELECT C1
00274 054100 100531 RCHC MMMM07,1 WRITE C1
00275 -162306 000537 MSM MMMM13,,6 MAKE CHANNEL B BUSY
00276 050000 100525 CLA MMMM03,1 READ COMMAND WORD
00277 060100 000531 STO MMMM07 SAVE READ COMMAND WORD
00300 052200 000500 XEC BBBB03 READ SELECT B1
00301 -054000 100525 RCHB MMMM03,1 READ B1
00302 -134107 000540 RUREDY PLT MMMM14,,7 IS CHANNEL C BUSY
00303 002000 000302 TRA *-1 YES
00304 002000 000237 TRA BBUSY NO, GO AND SEE IF CHANNEL B IS BUSY
*****
* BTRAP TEST *
*****
00305 060100 000477 TRAPB STO BBBB02 SAVE CONTENT OF ACCUMULATOR
00306 050000 000014 CLA 12
00307 076500 000024 LRS 20
00310 076000 000001 LBT 1 TEST FOR EOF, BIT 15
00311 002000 000313 TRA TEST2 TEST ANOTHER SWITCH
00312 002000 000360 TRA EOF AN END OF FILE HAS BEEN SENSED
00313 076300 000001 TEST2 LLS 1
00314 076000 000001 LBT 1 TEST FOR REDUNDANCY, BIT 16
00315 002000 000317 TRA TEST3 TEST ANOTHER SWITCH
00316 002000 000327 TRA REDUNB READ REDUNDANCY
00317 076300 000001 TEST3 LLS 1
00320 076000 000001 LBT 1 TEST FOR OP. COMP., BIT 17
00321 002000 000323 TRA TEST4 TEST ANOTHER SWITCH
00322 002000 000361 TRA RETRN A RECORD HAS BEEN SATISFACTORILY READ
00323 076500 000003 TEST4 LRS 3
00324 076000 000001 LBT 1 TEST FOR PARITY, BIT 14
00325 042000 000325 HPR * HALT, AN UNUSUAL CONDITION
00326 002000 000374 TRA BPARY PARITY CHECK, CPU
*****
* BTRAP *
*****

```

**Example 4 (Cont'd)**

```

00327 -064000 000503 REDUNB SCHB      BBBB06      STORE CH. B WORD COUNTER + ADDR. REG.
00330 -053400 400503          LXD      BBBB06,4    LOAD WORD COUNT INTO INDEX REG. 4
00331 335225 400343          TXH      RERED,4,14997 XFER TO RERED IF LESS THAN 3 WRDS. READ
00332 053400 400505          LXA      BBBB08,4
00333 076400 002221          BSR      1169      BACKSPACE
00334 006100 000334          TCOB     *
00335 200001 400337          TIX      RDB1,4,1
00336 042000 000336          HPR      *          PERMANENT READ REDUNDANCY
00337 063400 400505 RDB1  SXA      BBBB08,4
00340 052200 000500 XOXOXO XEC      BBBB03      SELECT B1
00341 -054000 000531          RCHB     MMMM07
00342 002000 000371          TRA      TTT
00343 053400 400510 RERED  LXA      BBBB11,4    10 TRYS WITH BMODE THEN FALLS THROUGH
00344 200001 400351          TIX      BMODE,4,1
00345 -162306 000504          MSM      BBBB07,,6
00346 050000 000501          CLA      BBBB04      SWITCH CONSTANT TO BINARY MODE
00347 060100 000500          STO      BBBB03      BINARY MODE
00350 002000 000340          TRA      XOXOXO
00351 063400 400510 BMODE  SXA      BBBB11,4
00352 076400 002221          BSR      1169
00353 006100 000353          TCOB     *
00354 050000 000502          CLA      BBBB05      SWITCH CONSTANT TO BCD MODE
00355 060100 000500          STO      BBBB03      BCD MODE
00356 -162307 000504          MSP      BBBB07,,7
00357 002000 000340          TRA      XOXOXO
00360 -162306 000541 EOF     MSM      MMMM15,,6    PERMITS ENTRANCE TO EOF ROUTINE
00361 050000 000507 RETRN  CLA      BBBB10      RESTORE
00362 060100 000505          STO      BBBB08      COUNTERS
00363 060100 000506          STO      BBBB09      FOR
00364 050000 000511          CLA      BBBB12      CHANNEL
00365 060100 000510          STO      BBBB11      B
00366 050000 000501          CLA      BBBB04      TRAP
00367 060100 000500          STO      BBBB03      ROUTINE
00370 -162307 000537          MSP      MMMM13,,7    FREES CHANNEL B
00371 050000 000477 TTT    CLA      BBBB02      RESTORE ACCUMULATOR
00372 076000 000014          RCT
00373 002060 000014          TRA*     12          RETURN TO MAIN ROUTINE
00374 053400 400506 BPARY  LXA      BBBB09,4
00375 076400 002221          BSR      1169      BACKSPACE
00376 006100 000376          TCOB     *
00377 200001 400401          TIX      RETRY,4,1
00400 042000 000400 PARITY HPR      *          HALT, EXCESSIVE PARITY CHECKS, CHANNEL B
00401 063400 400506 RETRY  SXA      BBBB09,4
00402 002000 000340          TRA      XOXOXO
*****
*          CTRAP TEST          *
*****
00403 060100 000522 TRAPC  STO      CCCC09      SAVE CONTENT OF ACCUMULATOR
00404 050000 000016          CLA      14          CHANNEL C TRAP STORE LOCATION
00405 076500 000023          LRS      19
00406 076000 000001          LBT      1          TEST FOR REDUNDANCY, BIT 16
00407 002000 000411          TRA      EXAM2      NO REDUNDANCY, TEST ANOTHER SWITCH
00410 002000 000421          TRA      REDON      REDUNDANT RECORD, CHANNEL C
00411 076300 000001 EXAM2  LLS      1
00412 076000 000001          LBT      1          TEST FOR OPERATION COMPLETE, BIT 17
00413 002000 000415          TRA      EXAM3      TEST ANOTHER SWITCH
00414 002000 000441          TRA      RETURN     OP. COMP., XFER + RESTORE COUNTERS
00415 076500 000003 EXAM3  LRS      3
00416 076000 000001          LBT      1          TEST FOR PARITY, BIT 14
00417 042000 000417          HPR      *          HALT, AN UNUSUAL CONDITION
00420 002000 000452          TRA      CPARY      PARITY CHECK, CPU
*****
*          CTRAP          *
*****
00421 053400 400516 REDON  LXA      CCCC05,4
00422 053400 200517          LXA      CCCC06,2
00423 076400 003221          BSR      1681
00424 006200 000424          TCOC     *
00425 200001 400430          TIX      REDON1,4,1
00426 200001 200434          TIX      REDON2,2,1
00427 042000 000427          HPR      *          PERMANENT WRITE REDUNDANCY
00430 063400 400516 REDON1  SXA      CCCC05,4
00431 052200 000513 CCC     XEC      CCCC02
00432 054100 000532          RCHC     MMMM08
00433 002000 000447          TRA      SSS
00434 063400 200516 REDON2  SXA      CCCC05,2
00435 063400 200517          SXA      CCCC06,2
00436 076610 003221 LLL    WBT      1681
00437 006200 000437          TCOC     *
00440 002000 000431          TRA      CCC

```

**Example 4 (Cont'd)**

```

00441 050000 000511 RETURN CLA      BBBB12
00442 060100 000516          STO      CCCC05
00443 060100 000517          STO      CCCC06
00444 060100 000520          STO      CCCC07
00445 060100 000521          STO      CCCC08
00446 -162307 000540          MSP      MMMM14,,7
00447 050000 000522 SSS      CLA      CCCC09      RESTORE ACCUMULATOR
00450 076000 000014          RCT
00451 002060 000016          TRA*     14
00452 053400 400520 CPARY  LX      CCCC07,4
00453 053400 200521          LX      CCCC08,2
00454 076400 003221          BSR     1681
00455 006200 000455          TCOC    *
00456 200001 400461          TIX     REDO,4,1
00457 200001 200463          TIX     REDU,2,1
00460 042000 000460          HPR     *      PERMANENT CPU PARITY CHECK
00461 063400 400520 REDO   SX      CCCC07,4
00462 002000 000431          TRA     CCC
00463 063400 200520 REDU   SX      CCCC07,2
00464 063400 200521          SX      CCCC08,2
00465 002000 000436          TRA     LLL
*****
*      WRITE TAPE MARK      *
*****
00466 -176000 000014 AFILE  ICT      INHIBIT ALL TRAPS
00467 077000 003221          WEF     1681      WRITE EOF ON C1
00470 006200 000470          TCOC    *      TEST READY STATUS OF CHANNEL C
00471 076000 000161          SWT     113      TEST S.S.W. 1-IF ON SKIP AN INSTRUCTION
00472 002000 000214          TRA     ANEW     PREPARE TO COPY ANOTHER FILE
00473 -077200 003221          RUN     1681      UNLOAD C1
00474 -077200 002221          RUN     1169     UNLOAD B1
00475 042000 000475          HPR     *      END OF JOB
*****
*      CONSTANTS-BTRAP TEST AND BTRAP      *
*****
00476 002000 000305 BBBB01 TRA      TRAPB      TO TRAP ROUTINE, CHANNEL B
00477 000000 000000 BBBB02 PZE      0      FOR RESTORING ACCUMULATOR
00500 076200 002221 BBBB03 RDS      1169     READ B1 IN BINARY
00501 076200 002221 BBBB04 RDS      1169     READ B1 IN BINARY
00502 076200 002201 BBBB05 RDS      1153     READ B1 IN BCD
00503 000000 000000 BBBB06 PZE      0      B CHANNEL WORD STORED HERE
00504 -000000 000000 BBBB07 MZE      0      IF PREFIX IS + THEN BINARY, IF - THEN BCD
00505 000000 000144 BBBB08 PZE      100     CHANNEL B REDUNDANCY, WILL TRY 99 TIMES
00506 000000 000144 BBBB09 PZE      100     CHANNEL B PARITY, WILL TRY 99 TIMES
00507 000000 000144 BBBB10 PZE      100     USED FOR RESTORING BBBB08 + BBBB09
00510 000000 000013 BBBB11 PZE      11      WILL TRY TO READ IN BCD 10 TIMES
00511 000000 000013 BBBB12 PZE      11      USED FOR RESTORING BBBB11,CCCC05,06,07,08
*****
*      CONSTANTS-CTRAP TEST AND CTRAP      *
*****
00512 002000 000403 CCCC01 TRA      TRAPC      TO TRAP ROUTINE, CHANNEL C
00513 076600 003221 CCCC02 WRS      1681     WRITE C1 IN BINARY
00514 076600 003221 CCCC03 WRS      1681     WRITE C1 IN BINARY
00515 076600 003201 CCCC04 WRS      1665     WRITE C1 IN BCD
00516 000000 000013 CCCC05 PZE      11      CHANNEL C REDUNDANCY, WILL TRY 55 TIMES
00517 000000 000013 CCCC06 PZE      11      ALLOWS WRITING OF BLANK TAPE UP TO 10 TIMES
00520 000000 000013 CCCC07 PZE      11      CHANNEL C PARITY, WILL TRY 55 TIMES
00521 000000 000013 CCCC08 PZE      11      ALLOWS WRITING OF BLANK TAPE UP TO 10 TIMES
00522 000000 000000 CCCC09 PZE      0      FOR RESTORING ACCUMULATOR
*****
*      CONSTANTS-MAIN ROUTINE      *
*****
00523 335230 000542 MMMM01 PTH      MMMM16,,15000 1ST READ COMMAND
00524 335230 035772 MMMM02 PTH      MMMM17,,15000 2ND READ COMMAND
00525 300000 000542 MMMM03 PTH      MMMM16,,0      1ST WRITE OUT BUFFER
00526 300000 035772 MMMM04 PTH      MMMM17,,0      2ND WRITE OUT BUFFER
00527 300000 000000 MMMM05 PTH      0      DETERMINES WRITE WORD COUNT
00530 300000 000000 MMMM06 PTH      0      DETERMINES WRITE WORD COUNT
00531 300000 000000 MMMM07 PTH      0      PRESENT READ COMMAND-FOR ERROR ROUTINE
00532 300000 000000 MMMM08 PTH      0      PRESENT WRITE COMMAND-FOR ERROR ROUTINE
00533 000000 000000 MMMM09 PZE      0      WHEN OP. COMP., CH. B WORD STORED HERE
00534 000000 000000 MMMM10 PZE      0      USED FOR DETERMINING WRITE WORD COUNT
00535 035230 000000 MMMM11 PZE      0,,15000     USED FOR DETERMINING WRITE WORD COUNT
00536 000002 000006 MMMM12 PZE      6,,2      MASKS CHANNEL B + C
00537 -000000 000000 MMMM13 MZE      0      USED FOR DETERMINING READY STATUS OF CH. B
00540 000000 000000 MMMM14 PZE      0      USED FOR DETERMINING READY STATUS OF CH. C
00541 000000 000000 MMMM15 PZE      0      USED FOR DETERMINING AN EOF
005420000          MMMM16 BSS      15000     1ST READ IN BUFFER
357720000          MMMM17 BSS      15000     2ND READ IN BUFFER
002170000          END      BEGIN

```

### Example 5

*Problem:* On the 1402, read cards with a 7-9 punch in card column 1 (column binary), edit for invalid BCD punching in the card, and punch the same information out, substituting blanks for invalid BCD characters. Column 1 will be blank on output cards, since any combination involving the 7-9 punch is invalid.

**IBM**

### Symbolic Coding Form

Problem		Date		Page		of	
Coder		Date		Page		of	
1	2	6	7	8	72	73	80
Location	Operation	Address, Tag, Decrement/Count	Comments	Identification			
	ORG	1024					
EDIT	RDS	664,,3	1230,,3. Read card binary.				
	RCHA	*+1					
	BCD	CARD,,27	26 words and four bytes.				
	AXT	27,1					
LOOP	CAL	CARD+27,1	Test in 12-bit bytes, three per word.				
	LGR	24					
	TSL	TEST					
	TNZ	*+3					
	SAC	CARD+27,1,0	Store zeros for invalid characters.				
	SAC	CARD+27,1,1					
	ZAC						
	LGL	12					
	TSL	TEST					
	TNZ	*+3					
	SAC	CARD+27,1,4					
	SAC	CARD+27,1,5					
	TIX	LOOP,1,1					
	WRS	665,,3	Write card binary.				
	RCHA	EDIT+2					
	TRA	EDIT					
GOOD	PXA	,2	Character test routine. Good return.				
	TRA*	TEST					
BAD	ZAC		Bad return.				
TEST	TRA	**	Entry and Exit.				

(continued on next page)

Example 5 (Cont'd)

IBM

Symbolic Coding Form

Problem		Date		Page		of	
Coder		Date		Page		of	
1	2	6	7	8	72	73	80
Location	Operation	Address, Tag, Decrement/Count	Comments	Identification			
	AXT	12, 6					
TEST 1	LBT		Count punches in character.				
	TIX	*+1, 2, 1					
	LGR	1					
	TIX	TEST 1, 4, 1					
	LGL	12	Shift character back.				
	TXL	TEST, 2, 1	One or less equals good.				
	TXH	BAD, 2, 3	Four or more equals bad.				
	TXH	TEST 3, 2, 2	Test 3 if triple punch.				
TEST 2	PAX	, 2	Double punch. Character to index 2.				
	TXH	TEST 4, 2, 2047	4000 equals a 12 zone.				
	TXH	TEST, 2, 511	11 or zero zone, all good.				
	TXH	BAD, 2, 255	Double digit punch. One punch is bad.				
	ANA	= 2	Must have 8 punch.				
	TZE	BAD					
	TXH	GOOD, 2, 3	Rest are good except 12-11.				
	TRA	BAD					
TEST 4	TXL	GOOD, 2, 3071	12 zone good except 12-11.				
	TRA	BAD					
TEST 3	PAX	, 2	Triple punch.				
	ARS	1					
	LBT		Must have 8 punch.				
	TRA	BAD					
	TXL	BAD, 2, 511	Must have a zone punch.				
	ANA	= 0176	Must have 2, 3, 4, 5, 6, or 7 punch.				

IBM

Symbolic Coding Form

Problem		Date		Page		of	
Coder		Date		Page		of	
1	2	6	7	8	72	73	80
Location	Operation	Address, Tag, Decrement/Count	Comments	Identification			
	TZE	BAD					
	ANA	= 0100					
	TZE	GOOD	Good if no 2 punch				
	TXL	GOOD, 2, 1023	0-2 punch good				
	TRA	BAD	11-2, 12-2 combinations are bad				
CARD	BSS	27					
	END	EDIT					



**Example 6**

Problem: Solve 
$$\frac{\sum_{i=1}^n X_i}{N_1} + \frac{\sum_{i=1}^n Y_i}{N_2}$$

Where  $X_i$  and  $Y_i$  are integers, and  $N_1$  and  $N_2$  are integers always greater than  $\sum X$  and  $\sum Y$  respectively.

**IBM**

**Symbolic Coding Form**

Problem							
Coder				Date	Page of		
1	2	3	4	5	6	7	8
Location			Operation	Address, Tag, Decrement/Count		Comments	Identification
			PXA				
			LDQ			= 0	
			AXT			0, 5	
			LXA			N, 2	
		ADD1	ADD			X1, 1	
			TXI			*+1, 1, -1	
			TI X			ADD1, 2, 1	
			DVP			N1	
			STQ			ANS	
			PXD				
			LDQ			= 0	
			XEC			ADD1-1	
		ADD2	ADD			Y1, 4	
			TXI			*+1, 4, -1	
			TI X			ADD2, 2, 1	
			DVP			N2	
			LLS			35	
			ADD			ANS	
			STQ			ANS	
			HPR				
		ANS	BSS			1	
		N	BSS			1	
		N1	BSS			1	
		N2	BSS			1	
		X1	BSS			NMAX	
		Y1	BSS			NMAX	
			END				

**Example 7**

Problem: Compute  $\sum_{i=1}^n (A_i + B_i) - C_i = D_i$   
 $i = 1, 2, \dots, n.$

**IBM**

**Symbolic Coding Form**

Problem								
Coder				Date		Page of		
•	Location			Operation	Address, Tag, Decrement/Count	Comments		Identification
1	2	4	7	8		72	73	80
				L X A	N, 1			
				P X D				
				A X T	O, 2			
				CLEAR ADD	A, 2			
				ADD	B, 2			
				S U B	C, 2			
				S T Ø	D, 2			
				ADD	C, 2			
				T X I	*+1, 2, -1			
				T I X	CLEAR, 1, 1			
				H P R				
				N B S S	1			
				A B S S	1			
				B B S S	1			
				C B S S	1			
				D B S S	1			
				END				

**Example 8**

Problem: Program  $X_i = V_i X_{i-1} + X_{i-2}$ ,  
 $i = 1, 2, \dots, n$ .

Where the values of  $X_{-1}$  and  $X_0$  are respectively minus one and zero. Consider all quantities as integers with no possibility of overflow.

**IBM**

**Symbolic Coding Form**

Problem				Date	Page	of
Coder						
Location		Operation	Address, Tag, Decrement/Count	Comments	Identification	
1	2	6	7	8	72	73
		LXA	N, 1			
		LXA	X+1, 2			
	LDQ	LDQ	R1, 2			
		MPY	X+1, 2			
		L L S	35			
		A D D	X, 2			
		S T Ø	X1, 2			
		T X I	*+1, 2, -1			
		T I X	LDQ, 1, 1			
		H P R				
	X	D E C	-1, 0			
	X 1	B S S	NMAX			
	R 1	B S S	NMAX			
	N	B S S	1			
		E N D				

**Example 9**

*Problem:* Compute  $K^x$  where  $x$  is a fraction by evaluating the series  $1 + \frac{X^2}{1!} + \frac{X}{2!} + \dots + \frac{X^k}{K!}$ , terminate when  $\left| \frac{X^k}{K!} \right| < .000000001$ .

**IBM**

Symbolic Coding Form

Problem							
Coder				Date	Page		of
•	Location		Operation	Address, Tag, Decrement/Count	Comments	Identification	
1	2	6	7	8		72	73
			STZ		K		
			CLA		SPEC1		
			STØ		SUM		
			STØ		TERM		
	LOOP		MSP		TERM		
			CLA		K		
			ADD		= 1		
			STØ		K		
			CLA		TERM		
			LRS		35		
			DVP		K		
			MPY		X		
			STØ		TERM		
			ADD		SUM		
			STØ		SUM		
			MSM		TERM		
			CLA		TEST		
			SUB		TERM		
			TMI		LOOP		
			HPR				
	K		BSS		1		
	SPEC1		DEC		1B2		
	SUM		BSS		SMAX		
	TERM		BSS		TMAX		
	TEST		DEC		1E-9B2		
	X		BSS		1		
			END				

Instructions for the 7040 and 7044 systems are offered in several options to satisfy different performance requirements. The basic set has been carefully selected to satisfactorily operate a low-compute requirement system application. The extended performance option enhances the computing and compiling ability by providing automatic indexing and logic, and character-handling operations. The single-precision floating-point option significantly improves performance on large number calculations and the double-precision floating-point option provides higher accuracy.

Indirect addressing ability is provided for all appropriate instructions, using the same method as with IBM 7090 and 7094 systems.

When the execution time of an instruction is variable, an instruction type number is included in the following instruction lists. To obtain the execution times in microseconds, multiply the number of cycles by the appropriate cycle time (2.5 or 8.0 microseconds). Both an alphabetic instruction list by option and a complete alphabetic list are included. The complete alphabetic list also indicates which central processing unit, data channel, and device indicators are set by execution of the instruction. For a detailed description of how the indicators are set, refer to the individual instruction description.

## Instruction Types

7040

7044

### *Type 1 – ALS, ARS, LGL, LGR, LLS, LRS, and RQL*

These instructions are executed in 1 cycle if the extent of the shift is six places or less. Each additional six-place shift or portion thereof requires  $\frac{1}{2}$  cycle.

These instructions are executed in 2 cycles if the extent of the shift is six places or less. Each additional six-place shift or portion thereof requires 1 cycle.

### *Type 2 – DVP*

This instruction is executed in  $7\frac{1}{2}$  cycles unless a divide check occurs, in which case it requires 2 cycles.

This instruction is executed in 20 cycles unless a divide check occurs, in which case it requires 3 cycles.

### *Type 3 – MPY*

This instruction is executed in 4 cycles if the MQ contains two or fewer ones. Each additional 6 ones or portion thereof in the MQ requires  $\frac{1}{2}$  cycle. If the content of Y is zero, the instruction is completed in 2 cycles.

This instruction is executed in 9 cycles if the MQ contains two or fewer ones. Each additional 6 ones or portion thereof in the MQ requires 1 cycle. If the content of Y is zero, the instruction is completed in 3 cycles.

7040

7044

### *Type 4 – VDP*

This instruction is executed in 2 cycles if the count is zero or one. Each additional two quotient positions or portion thereof requires  $\frac{1}{3}$  cycle.

This instruction is executed in 2 cycles if the count is zero. It requires 3 cycles if the count is one. Each additional two quotient positions or portion thereof requires 1 cycle.

### *Type 5 – VLM*

This instruction is executed in 2 cycles if the count is zero or one or if the content of Y is zero. Each additional six steps or portion thereof requires  $\frac{1}{2}$  cycle. To determine the number of additional steps: add the number of zeros to twice the number of ones in the low-order C bits of the MQ; then subtract one.

This instruction is executed in 2 cycles if the count is zero. It requires 3 cycles if the count is one or if the content of Y is zero. Each additional six steps or portion thereof requires 1 cycle. To determine the number of additional steps: add the number of zeros to twice the number of ones in the low-order C bits of the MQ; then subtract one.

### *Type 6 – FAD and FSB*

These instructions are executed in a minimum of  $2\frac{1}{2}$  cycles and a maximum of  $8\frac{1}{2}$  cycles. In determining average speed, a number of representative programs were traced. The times shown are based on an analysis of several million operands. Execution times greater than  $2\frac{1}{2}$  cycles are a result of shifting to equalize exponents before adding and to normalize the result after adding. Shifting requires  $\frac{1}{2}$  cycle for each six places or portion thereof.

These instructions are executed in a minimum of 4 cycles and a maximum of 23 cycles. In determining average speed, a number of representative programs were traced. The times shown are based on an analysis of several million operands. Execution times greater than 4 cycles are a result of shifting to equalize exponents before adding and to normalize the result after adding. Shifting requires one cycle for each six places or portion thereof.

### *Type 7 – FDP*

This instruction is executed in 7 cycles unless a divide check occurs, in which case it requires 2 cycles.

This instruction is executed in 18 cycles unless a divide check occurs, in which case it requires only 3 cycles.

### *Type 8 – FMP and UFM*

These instructions are executed in a minimum of  $3\frac{1}{2}$  cycles and a maximum of 5 cycles. If  $c(MQ)$  fraction is zero, it requires only 2 cycles.

These instructions are executed in a minimum of 8 cycles and a maximum of 12 cycles. If  $c(MQ)$  fraction is zero, it requires only 2 cycles.

### *Type 9 – UFA and UFS*

Execution time is the same as for type 6, except maximum is  $6\frac{1}{2}$  cycles due to un-normalized operation.

Execution time is the same as for type 6, except maximum is 16 cycles due to un-normalized operation.

**Type 10 – DFAD, DFSB**

These instructions are executed in a minimum of 4 cycles and a maximum of 11 cycles. The longer times are a result of shifting, as explained in Type 6.

These instructions are executed in a minimum of 7 cycles and a maximum of 28 cycles. The longer times are a result of shifting, as explained in Type 6.

**Type 11 – DFMP**

This instruction is executed in a maximum of 13 $\frac{2}{3}$  cycles. If c(AC) and c(MQ) are zero, the instruction requires 3 cycles.

This instruction is executed in a maximum of 36 cycles. If c(AC) and c(MQ) are zero, the instruction requires 3 cycles.

**Type 12 – DFDP**

This instruction is executed in a maximum of 18 $\frac{1}{3}$  cycles, and a minimum of 17 cycles. If a divide check occurs, this instruction may require as few as 3 cycles.

This instruction is executed in a maximum of 50 cycles, and a minimum of 46 cycles. If a divide check occurs, this instruction may require as few as 4 cycles.

**Type 13 – BSR, ETT, PRD, PWR, RDS, REW, RUN, SEN, WBT, WEF, and WRS**

These instructions are executed in the times given if the channel is not busy and the device selected is ready and not busy. Otherwise, execution is delayed until these conditions do exist. If the channel is not busy and the on-line 1401 is selected, a programmed response is required from the 1401 before these instructions can complete execution.

These instructions are executed in the times given if the channel is not busy and the device selected is ready and not busy. Otherwise, execution is delayed until these conditions do exist. If the channel is not busy and the on-line 1401 is selected, a programmed response is required from the 1401 before these instructions can complete execution.

**Type 14 – BSR, REW, RUN, and WEF**

These instructions complete execution in the times given, but the channel remains busy for the duration of the back-space or write end of file. The channel is busy on rewind instructions only long enough to pick relays in the tape unit.

These instructions complete execution in the times given, but the channel remains busy for the duration of the back-space or write end of file. The channel is busy on rewind instructions only long enough to pick relays in the tape unit.

**Type 15 – VMA**

This instruction is executed in 2 cycles if the count is zero or one. Each additional 6 steps or portion thereof requires  $\frac{1}{3}$  cycle. To determine the number of additional steps add the number of "zeros" to twice the number of "ones" in the low order C bits of the MQ, then subtract one.

This instruction is executed in 2 cycles if the count is zero. Three cycles are required if the count is one. Each additional 6 steps or portion thereof requires 1 cycle. To determine the number of additional steps add the number of "zeros" to twice the number of "ones" in the low order C bits of the MQ, then subtract one.

**Alphabetic Instruction List — By Option**

INST	OP CODE	AVERAGE CYCLES		TYPE	PAGE
		7040	7044		
<b>Basic Instruction Set</b>					
ACL	+0361	2	2		62
ADD	+0400	2	2		45

INST	OP CODE	AVERAGE CYCLES		TYPE	PAGE
		7040	7044		
ALS	+0767	2	4	1	49
ANA	-0320	2	2		65
ARS	+0771	2	4	1	50
CAL	-0500	2	2		62
CAP	-1510	2	2		63
CAS	+0340	2	3		54
CHS	+0760 .002	1	2		54
CLA	+0500	2	2		45
CLS	+0502	2	2		45
COM	+0760 .006	1	2		61
DCT	+0760 .012	1	2		52
DVP	+0221	7 $\frac{2}{3}$	20	2	48
ENK	+0760 .004	1	2		54
HPR	+0420	1	2		52
LAS	-0340	2	3		63
LBT	+0760 .001	1	2		52
LDQ	+0560	2	2		46
LGL	-0763	2	4	1	62
LGR	-0765	2	4	1	63
LLS	+0763	2	4	1	50
LRS	+0765	2	4	1	51
MPY	+0200	5	12	3	46
ORA	-0501	2	2		65
PBT	-0760 .001	1	2		52
RQI	-0773	2	4	1	51
SLP	-1612	2	2		63
SLW	+0602	2	2		63
SSP	+0760 .003	1	2		54
STA	+0621	3	3		58
STD	+0622	3	3		58
STL	-0625	3	3		123
STO	+0601	2	2		45
STQ	-0600	2	2		46
STR	-1000	2	2		123
STZ	+0600	2	2		48
SUB	+0402	2	2		45
SWT	+0760 .16x	1	2		52
TMI	-0120	1	1		53
TNZ	-0100	1	1		53
TOV	+0140	1	1		53
TPL	+0120	1	1		53
TRA	+0020	1	1		53
TRP	-1165	1	1		122
TRT	-1164	1	1		122
TSL	-1627	3	3		123
TZE	+0100	1	1		53
VDP	+0225	5	10	4	49
VLM	+0204	4	9	5	48
VMA	-1204	—	—	15	48
XEC	+0522	1	1		52

**Extended Performance Set**

AXT	+0774	1	1		58
CCS	-1341	2	3		67
LAC	+0535	2	2		58
LDC	-0535	2	2		58
LXA	+0534	2	2		58
LXD	-0534	2	2		58
MIT	-1341	2	3		54
MSM	-1623	3	3		54
MSP	-1623	3	3		54
PAC	+0737	1	2		58





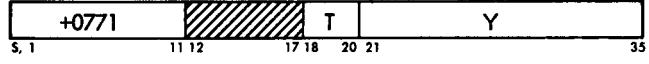

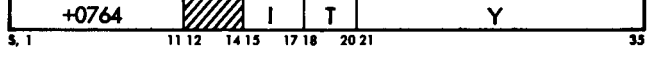

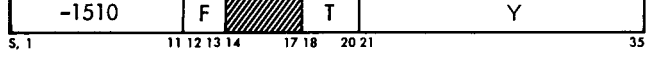
INST	OP CODE	AVERAGE CYCLES		TYPE	PAGE	INST	OP CODE	AVERAGE CYCLES		TYPE	PAGE
		7040	7044					7040	7044		
PAX	+0734	1	2		58	PSLD	-0665	2	3		113
PCS	-1505	2	2		67	PSLE	+0666	2	3		113
PDC	-0737	1	2		58	SSLB	-0660	2	2		113
PDX	-0734	1	2		58	SSLC	+0661	2	2		113
PLT	-1341	2	3		54	SSLD	-0661	2	2		113
PXA	+0754	1	2		58	SSLE	+0662	2	2		113
PXD	-0754	1	2		58						
SAC	-1623	3	3		67	<i>Input/Output Instructions</i>					
SXA	+0634	3	3		59	BSR	+0764	2	4	13, 14	93
SXD	-0634	3	3		59	CTR	-1766	1	2		105
TIX	+2000	1	2		59	ENB	+0564	2	2		122
TMT	-1704	1+2N	2+2N		67	ETT	-0760.x2xx	1	2	13	96
TNX	-2000	1	2		60	ICT	-1760.014	1	2		122
TSX	+0074	1	2		62	IOT	+0760.005	1	2		97
TXH	+3000	1	2		60	PRD	-1762	2	4	13	103
TXI	+1000	1	2		60	PWR	-1766	2	4	13	103
TXL	-3000	1	2		60						
<i>Single-Precision Floating-Point Set</i>											
FAD	+0300	3	5½	6	71	RCHA	+0540	2	2		91
FDP	+0241	7	18	7	73	RCT	+0760.014	1	2		122
FMP	+0260	4½	10	8	73	RDC	+0760.x352	1	2		97
FSB	+0302	3	5½	6	73	RDS	+0762	2	4	13	90
UFA	-0300	2¾	5	9	73	REW	+0772	2	4	13, 14	96
UFM	-0260	4½	10	8	73	RUN	-0772	2	4	13, 14	96
UFS	-0302	2¾	5	9	73	SCHA	+0640	2	2		97
<i>Double-Precision Floating-Point Set</i>											
DFAD	+0301	4½	8½	10	74	SEN	-1762	1	2	13	104
DFDP	-0241	17¾	48	12	74	TCOA	+0060	1	2		92
DFMP	+0261	12	31	11	74	TDOA	-1060	1	2		104
DFSB	+0303	4½	8½	10	74	TEF	+0030	1	2		95
<i>Memory Protect Set</i>											
RPM	-1004	2	2		114	TRC	+0022	1	2		92
SPM	-1160	1	1		114	WBT	+0766	2	4	13	94
<i>Direct Data Set</i>											
PSLB	-0664	2	3		113	WEF	+0770	2	4	13, 14	95
PSLC	+0665	2	3		113	WRS	+0766	2	4	13	90
<i>1401 Option Instructions</i>											
						SLFA	-1760	1	2		115
						SLNA	-1760	1	2		115

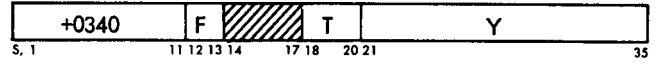
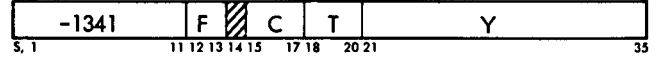


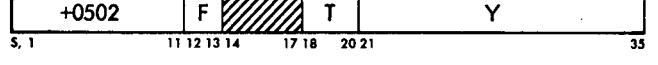
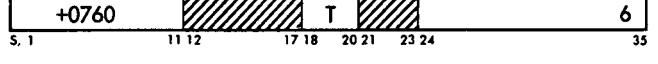
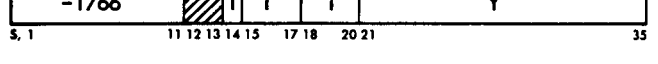
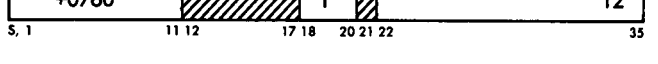
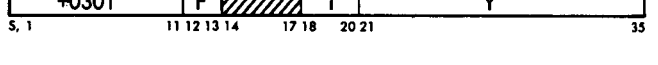
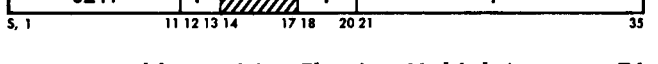
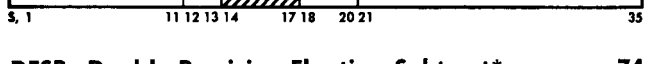
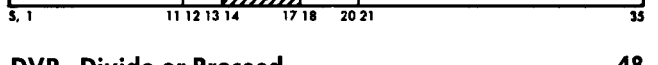
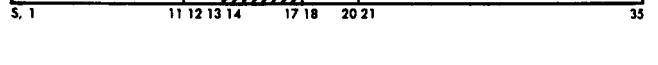
## Appendix B. Instruction List — Alphabetic Order with Formats

Symbols used with the instruction formats are:

F	Indirect Addressing Flag Field
C	Count Field
I	Channel A I/O Device Adapter Field
S	Card Punch Stacker Select Character
B	I/O Device Busy Status Character
M	I/O Device Input/Output Buffer Select Character
T	Index Register Tag Field
Y	Operand Designation Field

Instructions are listed in alphabetic order without regard to optional features. An asterisk (\*) following the instruction name designates an optional instruction. Operation codes are shown in octal notation.

MNEMONIC AND NAME	PAGE
<b>ACL—Add and Carry Logical Word</b> 62	
	
<b>ADD—Add</b> 45	
	
<b>ALS—Accumulator Left Shift</b> 49	
	
<b>ANA—And to Accumulator</b> 65	
	
<b>ARS—Accumulator Right Shift</b> 50	
	
<b>AXT—Address to Index True*</b> 58	
	
<b>BSR—Backspace Record</b> 93	
	
<b>CAL—Clear and Add Logical Word</b> 62	
	
<b>CAP—Clear and Add Logical Word/Parity</b> 63	
	

MNEMONIC AND NAME	PAGE
<b>CAS—Compare Accumulator with Storage</b> 54	
	
<b>CCS—Compare Character with Storage*</b> 67	
	
<b>CHS—Change Sign</b> 54	
	
<b>CLA—Clear and Add</b> 45	
	
<b>CLS—Clear and Subtract</b> 45	
	
<b>COM—Complement Magnitude</b> 61	
	
<b>CTR—Control Select</b> 105	
	
<b>DCT—Divide Check Test</b> 52	
	
<b>DFAD—Double Precision Floating Add*</b> 74	
	
<b>DFDP—Double Precision Divide or Proceed*</b> 74	
	
<b>DFMP—Double Precision Floating Multiply*</b> 74	
	
<b>DFSB—Double Precision Floating Subtract*</b> 74	
	
<b>DVP—Divide or Proceed</b> 48	
	



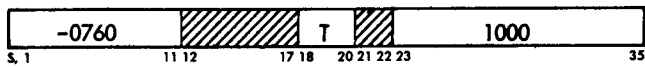
**ENB—Enable from Y 122**



**ENK—Enter Keys 54**



**ETTA—End of Tape Test, Channel A 96**



- ETTB — 0760 2000
- ETTC — 0760 3000
- ETTD — 0760 4000
- ETTE — 0760 5000

**FAD—Floating Point Add\* 71**



**FDP—Floating Divide or Proceed\* 73**



**FMP—Floating Point Multiply\* 73**



**FSB—Floating Point Subtract\* 73**



**HPR—Halt and Proceed 52**



**ICT—Inhibit Channel Traps 122**



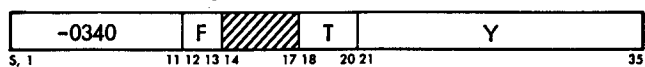
**IOT—Input/Output Check Test 97**



**LAC—Load-Complement of Address in Index\* 58**



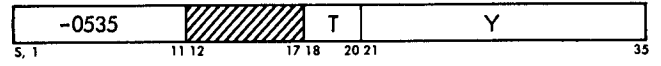
**LAS—Logical Compare Accumulator with Storage 63**



**LBT—Low Bit Test 52**



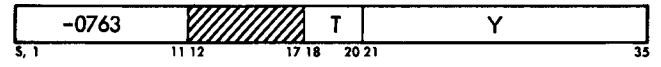
**LDC—Load Complement of Decrement in Index\* 58**



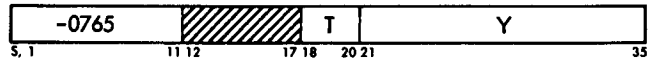
**LDQ—Load Multiplier-Quotient 46**



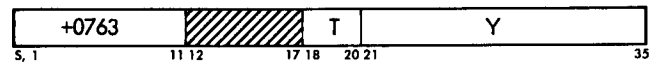
**LGL—Logical Left Shift 62**



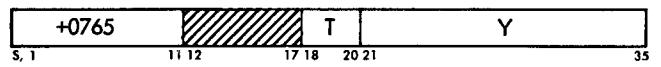
**LGR—Logical Right Shift 63**



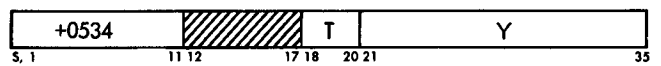
**LLS—Long Left Shift 50**



**LRS—Long Right Shift 51**



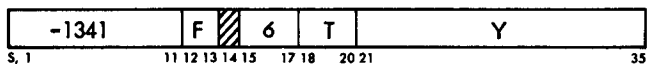
**LXA—Load Index from Address\* 58**



**LXD—Load Index from Decrement\* 58**



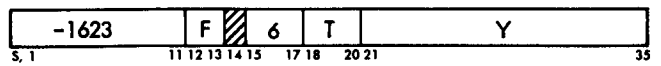
**MIT—Storage Minus Test\* 54**



**MPY—Multiply 46**



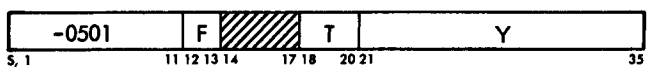
**MSM—Make Storage Sign Minus\* 54**



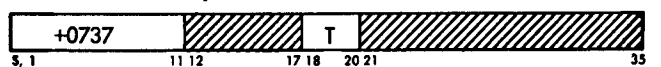
**MSP—Make Storage Sign Plus\* 54**



**ORA—Or to Accumulator 65**



**PAC—Place Complement of Index in Address\* 58**



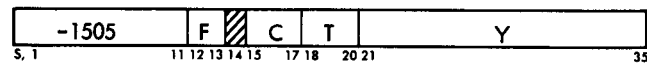
**PAX—Place Address in Index\*** 58



**PBT—P Bit Test** 52



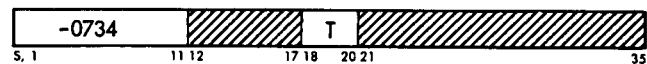
**PCS—Place Character from Storage\*** 67



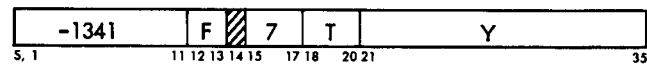
**PDC—Place Complement of Decrement in Index\*** 58



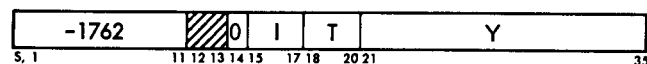
**PDX—Place Decrement in Index\*** 58



**PLT—Storage Plus Test\*** 54



**PRD—Prepare to Read** 103



**PSLB—Present Sense Lines, Channel B\*** 113



- PSLC + 0665
- PSLD - 0665
- PSLE + 0666

**PWR—Prepare to Write** 103



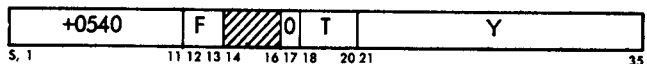
**PXA—Place Index in Address\*** 58



**PXD—Place Index in Decrement\*** 58



**RCHA—Reset and Load Channel A** 91



- RCHB - 0540
- RCHC + 0541
- RCHD - 0541
- RCHE + 0542

**RCT—Restore Channel Traps** 122

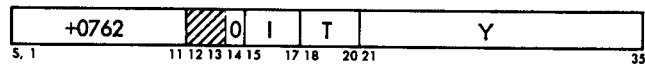


**RDCA—Reset Data Channel A** 97

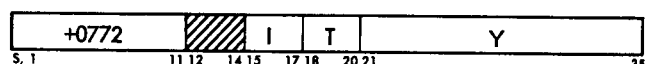


- RDCB + 0760 2352
- RDCC + 0760 3352
- RDCD + 0760 4352
- RDCE + 0760 5352

**RDS—Read Select** 90



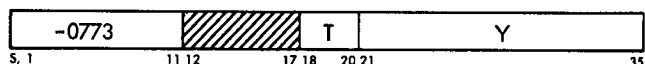
**REW—Rewind** 96



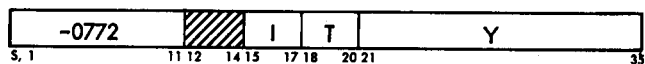
**RPM—Release Protect Mode\*** 114



**RQL—Rotate Quotient Left** 51



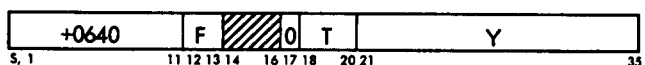
**RUN—Rewind and Unload** 96



**SAC—Store Accumulator Character\*** 67

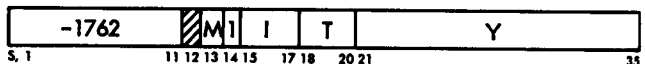


**SCHA—Store Channel A** 97

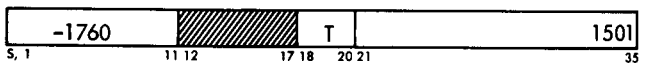


- SCHB - 0640
- SCHC + 0641
- SCHD - 0641
- SCHE + 0642

**SEN—Sense Select** 104



**SLFA—Status Line Off, Channel A\*** 115



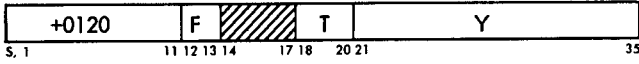
**SLNA—Status Line On, Channel A\*** 115



MNEMONIC AND NAME	PAGE
<b>SLP—Store Logical Word with Parity</b> 63	
<b>SLW—Store Logical Word</b> 63	
<b>SPM—Set Protect Mode*</b> 114	
<b>SSLB—Store Sense Lines, Channel B*</b> 113	
SSLC + 0661 SSLD - 0661 SSLE + 0662	
<b>SSP—Set Sign Plus</b> 54	
<b>STA—Store Address</b> 58	
<b>STD—Store Decrement</b> 58	
<b>STL—Store Instruction Counter</b> 123	
<b>STO—Store Accumulator</b> 45	
<b>STQ—Store Multiplier-Quotient</b> 46	
<b>STR—Store Location and Trap</b> 123	
<b>STZ—Store Zero</b> 48	
<b>SUB—Subtract</b> 45	

MNEMONIC AND NAME	PAGE
<b>SWT—Sense Switch Test</b> 52	
<b>SXA—Store Index in Address*</b> 59	
<b>SXD—Store Index in Decrement*</b> 59	
<b>TCOA—Transfer on Channel A in Operation</b> 92	
TCOB + 0061 TCOC + 0062 TCOD + 0063 TCOE + 0064	
<b>TDOA—Transfer on Device in Operation, Channel A</b> 104	
<b>TEFA—Transfer on End of File, Channel A</b> 95	
TEFB - 0030 TEFC + 0031 TEFD - 0031 TEFE + 0032	
<b>TIX—Transfer on Index*</b> 59	
<b>TMI—Transfer on Minus</b> 53	
<b>TMT—Transmit*</b> 67	
<b>TNX—Transfer on No Index*</b> 60	
<b>TNZ—Transfer on No Zero</b> 53	
<b>TOV—Transfer on Overflow</b> 53	

**TPL—Transfer on Plus 53**



**TRA—Transfer 53**



**TRCA—Transfer on Redundancy Check, Channel A 92**

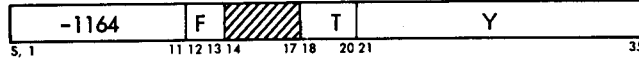


TRCB - 0022  
 TRCC + 0024  
 TRCD - 0024  
 TRCE + 0026

**TRP—Transfer and Restore Parity and Traps 122**



**TRT—Transfer and Restore Traps 122**



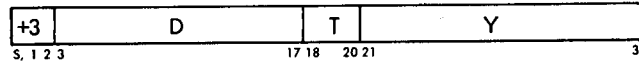
**TSL—Transfer and Store Instruction Counter 123**



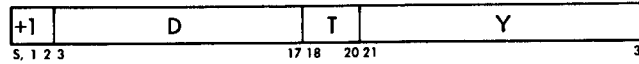
**TSX—Transfer and Set Index\* 62**



**TXH—Transfer on Index High\* 60**



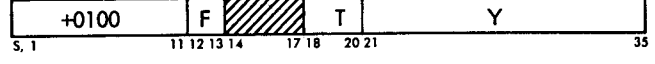
**TXI—Transfer with Index Incremented\* 60**



**TXL—Transfer on Index Low\* 60**



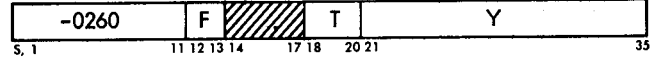
**TZE—Transfer on Zero 53**



**UFA—Unnormalized Floating Add\* 73**



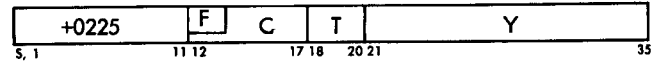
**UFM—Unnormalized Floating Multiply\* 73**



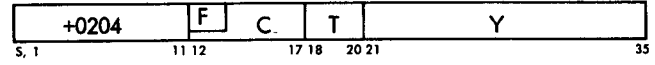
**UFS—Unnormalized Floating Subtract\* 73**



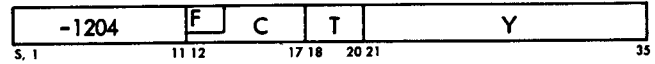
**VDP—Variable Divide or Proceed 49**



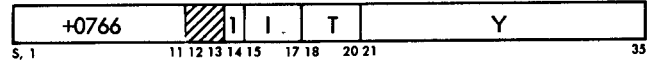
**VLM—Variable Length Multiply 48**



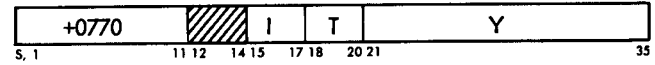
**VMA—Variable Length Multiply/Accumulate 48**



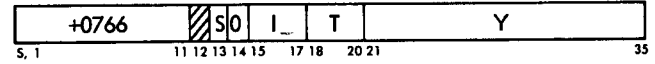
**WBT—Write Blank Tape 94**



**WEF—Write End of File 95**



**WRS—Write Select 90**



**XEC—Execute 52**



## Appendix C. Powers of Two Table

$2^n$	$n$	$2^{-n}$
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125











## Appendix E. Octal-Decimal Fraction Conversion Table

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000	.000000	.100	.125000	.200	.250000	.300	.375000
.001	.001953	.101	.126953	.201	.251953	.301	.376953
.002	.003906	.102	.128906	.202	.253906	.302	.378906
.003	.005859	.103	.130859	.203	.255859	.303	.380859
.004	.007812	.104	.132812	.204	.257812	.304	.382812
.005	.009765	.105	.134765	.205	.259765	.305	.384765
.006	.011718	.106	.136718	.206	.261718	.306	.386718
.007	.013671	.107	.138671	.207	.263671	.307	.388671
.010	.015625	.110	.140625	.210	.265625	.310	.390625
.011	.017578	.111	.142578	.211	.267578	.311	.392578
.012	.019531	.112	.144531	.212	.269531	.312	.394531
.013	.021484	.113	.146484	.213	.271484	.313	.396484
.014	.023437	.114	.148437	.214	.273437	.314	.398437
.015	.025390	.115	.150390	.215	.275390	.315	.400390
.016	.027343	.116	.152343	.216	.277343	.316	.402343
.017	.029296	.117	.154296	.217	.279296	.317	.404296
.020	.031250	.120	.156250	.220	.281250	.320	.406250
.021	.033203	.121	.158203	.221	.283203	.321	.408203
.022	.035156	.122	.160156	.222	.285156	.322	.410156
.023	.037109	.123	.162109	.223	.287109	.323	.412109
.024	.039062	.124	.164062	.224	.289062	.324	.414062
.025	.041015	.125	.166015	.225	.291015	.325	.416015
.026	.042968	.126	.167968	.226	.292968	.326	.417968
.027	.044921	.127	.169921	.227	.294921	.327	.419921
.030	.046875	.130	.171875	.230	.296875	.330	.421875
.031	.048828	.131	.173828	.231	.298828	.331	.423828
.032	.050781	.132	.175781	.232	.300781	.332	.425781
.033	.052734	.133	.177734	.233	.302734	.333	.427734
.034	.054687	.134	.179687	.234	.304687	.334	.429687
.035	.056640	.135	.181640	.235	.306640	.335	.431640
.036	.058593	.136	.183593	.236	.308593	.336	.433593
.037	.060546	.137	.185546	.237	.310546	.337	.435546
.040	.062500	.140	.187500	.240	.312500	.340	.437500
.041	.064453	.141	.189453	.241	.314453	.341	.439453
.042	.066406	.142	.191406	.242	.316406	.342	.441406
.043	.068359	.143	.193359	.243	.318359	.343	.443359
.044	.070312	.144	.195312	.244	.320312	.344	.445312
.045	.072265	.145	.197265	.245	.322265	.345	.447265
.046	.074218	.146	.199218	.246	.324218	.346	.449218
.047	.076171	.147	.201171	.247	.326171	.347	.451171
.050	.078125	.150	.203125	.250	.328125	.350	.453125
.051	.080078	.151	.205078	.251	.330078	.351	.455078
.052	.082031	.152	.207031	.252	.332031	.352	.457031
.053	.083984	.153	.208984	.253	.333984	.353	.458984
.054	.085937	.154	.210937	.254	.335937	.354	.460937
.055	.087890	.155	.212890	.255	.337890	.355	.462890
.056	.089843	.156	.214843	.256	.339843	.356	.464843
.057	.091796	.157	.216796	.257	.341796	.357	.466796
.060	.093750	.160	.218750	.260	.343750	.360	.468750
.061	.095703	.161	.220703	.261	.345703	.361	.470703
.062	.097656	.162	.222656	.262	.347656	.362	.472656
.063	.099609	.163	.224609	.263	.349609	.363	.474609
.064	.101562	.164	.226562	.264	.351562	.364	.476562
.065	.103515	.165	.228515	.265	.353515	.365	.478515
.066	.105468	.166	.230468	.266	.355468	.366	.480468
.067	.107421	.167	.232421	.267	.357421	.367	.482421
.070	.109375	.170	.234375	.270	.359375	.370	.484375
.071	.111328	.171	.236328	.271	.361328	.371	.486328
.072	.113281	.172	.238281	.272	.363281	.372	.488281
.073	.115234	.173	.240234	.273	.365234	.373	.490234
.074	.117187	.174	.242187	.274	.367187	.374	.492187
.075	.119140	.175	.244140	.275	.369140	.375	.494140
.076	.121093	.176	.246093	.276	.371093	.376	.496093
.077	.123046	.177	.248046	.277	.373046	.377	.498046

## Octal-Decimal Fraction Conversion Table

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000000	.000000	.000100	.000244	.000200	.000488	.000300	.000732
.000001	.000003	.000101	.000247	.000201	.000492	.000301	.000736
.000002	.000007	.000102	.000251	.000202	.000495	.000302	.000740
.000003	.000011	.000103	.000255	.000203	.000499	.000303	.000743
.000004	.000015	.000104	.000259	.000204	.000503	.000304	.000747
.000005	.000019	.000105	.000263	.000205	.000507	.000305	.000751
.000006	.000022	.000106	.000267	.000206	.000511	.000306	.000755
.000007	.000026	.000107	.000270	.000207	.000514	.000307	.000759
.000010	.000030	.000110	.000274	.000210	.000518	.000310	.000762
.000011	.000034	.000111	.000278	.000211	.000522	.000311	.000766
.000012	.000038	.000112	.000282	.000212	.000526	.000312	.000770
.000013	.000041	.000113	.000286	.000213	.000530	.000313	.000774
.000014	.000045	.000114	.000289	.000214	.000534	.000314	.000778
.000015	.000049	.000115	.000293	.000215	.000537	.000315	.000782
.000016	.000053	.000116	.000297	.000216	.000541	.000316	.000785
.000017	.000057	.000117	.000301	.000217	.000545	.000317	.000789
.000020	.000061	.000120	.000305	.000220	.000549	.000320	.000793
.000021	.000064	.000121	.000308	.000221	.000553	.000321	.000797
.000022	.000068	.000122	.000312	.000222	.000556	.000322	.000801
.000023	.000072	.000123	.000316	.000223	.000560	.000323	.000805
.000024	.000076	.000124	.000320	.000224	.000564	.000324	.000808
.000025	.000080	.000125	.000324	.000225	.000568	.000325	.000812
.000026	.000083	.000126	.000328	.000226	.000572	.000326	.000816
.000027	.000087	.000127	.000331	.000227	.000576	.000327	.000820
.000030	.000091	.000130	.000335	.000230	.000579	.000330	.000823
.000031	.000095	.000131	.000339	.000231	.000583	.000331	.000827
.000032	.000099	.000132	.000343	.000232	.000587	.000332	.000831
.000033	.000102	.000133	.000347	.000233	.000591	.000333	.000835
.000034	.000106	.000134	.000350	.000234	.000595	.000334	.000839
.000035	.000110	.000135	.000354	.000235	.000598	.000335	.000843
.000036	.000114	.000136	.000358	.000236	.000602	.000336	.000846
.000037	.000118	.000137	.000362	.000237	.000606	.000337	.000850
.000040	.000122	.000140	.000366	.000240	.000610	.000340	.000854
.000041	.000125	.000141	.000370	.000241	.000614	.000341	.000858
.000042	.000129	.000142	.000373	.000242	.000617	.000342	.000862
.000043	.000133	.000143	.000377	.000243	.000621	.000343	.000865
.000044	.000137	.000144	.000381	.000244	.000625	.000344	.000869
.000045	.000141	.000145	.000385	.000245	.000629	.000345	.000873
.000046	.000144	.000146	.000389	.000246	.000633	.000346	.000877
.000047	.000148	.000147	.000392	.000247	.000637	.000347	.000881
.000050	.000152	.000150	.000396	.000250	.000640	.000350	.000885
.000051	.000156	.000151	.000400	.000251	.000644	.000351	.000888
.000052	.000160	.000152	.000404	.000252	.000648	.000352	.000892
.000053	.000164	.000153	.000408	.000253	.000652	.000353	.000896
.000054	.000167	.000154	.000411	.000254	.000656	.000354	.000900
.000055	.000171	.000155	.000415	.000255	.000659	.000355	.000904
.000056	.000175	.000156	.000419	.000256	.000663	.000356	.000907
.000057	.000179	.000157	.000423	.000257	.000667	.000357	.000911
.000060	.000183	.000160	.000427	.000260	.000671	.000360	.000915
.000061	.000186	.000161	.000431	.000261	.000675	.000361	.000919
.000062	.000190	.000162	.000434	.000262	.000679	.000362	.000923
.000063	.000194	.000163	.000438	.000263	.000682	.000363	.000926
.000064	.000198	.000164	.000442	.000264	.000686	.000364	.000930
.000065	.000202	.000165	.000446	.000265	.000690	.000365	.000934
.000066	.000205	.000166	.000450	.000266	.000694	.000366	.000938
.000067	.000209	.000167	.000453	.000267	.000698	.000367	.000942
.000070	.000213	.000170	.000457	.000270	.000701	.000370	.000946
.000071	.000217	.000171	.000461	.000271	.000705	.000371	.000949
.000072	.000221	.000172	.000465	.000272	.000709	.000372	.000953
.000073	.000225	.000173	.000469	.000273	.000713	.000373	.000957
.000074	.000228	.000174	.000473	.000274	.000717	.000374	.000961
.000075	.000232	.000175	.000476	.000275	.000720	.000375	.000965
.000076	.000236	.000176	.000480	.000276	.000724	.000376	.000968
.000077	.000240	.000177	.000484	.000277	.000728	.000377	.000972

### Octal-Decimal Fraction Conversion Table

OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.	OCTAL	DEC.
.000400	.000976	.000500	.001220	.000600	.001464	.000700	.001708
.000401	.000980	.000501	.001224	.000601	.001468	.000701	.001712
.000402	.000984	.000502	.001228	.000602	.001472	.000702	.001716
.000403	.000988	.000503	.001232	.000603	.001476	.000703	.001720
.000404	.000991	.000504	.001235	.000604	.001480	.000704	.001724
.000405	.000995	.000505	.001239	.000605	.001483	.000705	.001728
.000406	.000999	.000506	.001243	.000606	.001487	.000706	.001731
.000407	.001003	.000507	.001247	.000607	.001491	.000707	.001735
.000410	.001007	.000510	.001251	.000610	.001495	.000710	.001739
.000411	.001010	.000511	.001255	.000611	.001499	.000711	.001743
.000412	.001014	.000512	.001258	.000612	.001502	.000712	.001747
.000413	.001018	.000513	.001262	.000613	.001506	.000713	.001750
.000414	.001022	.000514	.001266	.000614	.001510	.000714	.001754
.000415	.001026	.000515	.001270	.000615	.001514	.000715	.001758
.000416	.001029	.000516	.001274	.000616	.001518	.000716	.001762
.000417	.001033	.000517	.001277	.000617	.001522	.000717	.001766
.000420	.001037	.000520	.001281	.000620	.001525	.000720	.001770
.000421	.001041	.000521	.001285	.000621	.001529	.000721	.001773
.000422	.001045	.000522	.001289	.000622	.001533	.000722	.001777
.000423	.001049	.000523	.001293	.000623	.001537	.000723	.001781
.000424	.001052	.000524	.001296	.000624	.001541	.000724	.001785
.000425	.001056	.000525	.001300	.000625	.001544	.000725	.001789
.000426	.001060	.000526	.001304	.000626	.001548	.000726	.001792
.000427	.001064	.000527	.001308	.000627	.001552	.000727	.001796
.000430	.001068	.000530	.001312	.000630	.001556	.000730	.001800
.000431	.001071	.000531	.001316	.000631	.001560	.000731	.001804
.000432	.001075	.000532	.001319	.000632	.001564	.000732	.001808
.000433	.001079	.000533	.001323	.000633	.001567	.000733	.001811
.000434	.001083	.000534	.001327	.000634	.001571	.000734	.001815
.000435	.001087	.000535	.001331	.000635	.001575	.000735	.001819
.000436	.001091	.000536	.001335	.000636	.001579	.000736	.001823
.000437	.001094	.000537	.001338	.000637	.001583	.000737	.001827
.000440	.001098	.000540	.001342	.000640	.001586	.000740	.001831
.000441	.001102	.000541	.001346	.000641	.001590	.000741	.001834
.000442	.001106	.000542	.001350	.000642	.001594	.000742	.001838
.000443	.001110	.000543	.001354	.000643	.001598	.000743	.001842
.000444	.001113	.000544	.001358	.000644	.001602	.000744	.001846
.000445	.001117	.000545	.001361	.000645	.001605	.000745	.001850
.000446	.001121	.000546	.001365	.000646	.001609	.000746	.001853
.000447	.001125	.000547	.001369	.000647	.001613	.000747	.001857
.000450	.001129	.000550	.001373	.000650	.001617	.000750	.001861
.000451	.001132	.000551	.001377	.000651	.001621	.000751	.001865
.000452	.001136	.000552	.001380	.000652	.001625	.000752	.001869
.000453	.001140	.000553	.001384	.000653	.001628	.000753	.001873
.000454	.001144	.000554	.001388	.000654	.001632	.000754	.001876
.000455	.001148	.000555	.001392	.000655	.001636	.000755	.001880
.000456	.001152	.000556	.001396	.000656	.001640	.000756	.001884
.000457	.001155	.000557	.001399	.000657	.001644	.000757	.001888
.000460	.001159	.000560	.001403	.000660	.001647	.000760	.001892
.000461	.001163	.000561	.001407	.000661	.001651	.000761	.001895
.000462	.001167	.000562	.001411	.000662	.001655	.000762	.001899
.000463	.001171	.000563	.001415	.000663	.001659	.000763	.001903
.000464	.001174	.000564	.001419	.000664	.001663	.000764	.001907
.000465	.001178	.000565	.001422	.000665	.001667	.000765	.001911
.000466	.001182	.000566	.001426	.000666	.001670	.000766	.001914
.000467	.001186	.000567	.001430	.000667	.001674	.000767	.001918
.000470	.001190	.000570	.001434	.000670	.001678	.000770	.001922
.000471	.001194	.000571	.001438	.000671	.001682	.000771	.001926
.000472	.001197	.000572	.001441	.000672	.001686	.000772	.001930
.000473	.001201	.000573	.001445	.000673	.001689	.000773	.001934
.000474	.001205	.000574	.001449	.000674	.001693	.000774	.001937
.000475	.001209	.000575	.001453	.000675	.001697	.000775	.001941
.000476	.001213	.000576	.001457	.000676	.001701	.000776	.001945
.000477	.001216	.000577	.001461	.000677	.001705	.000777	.001949

## Appendix F. Scaling for Fixed-Point Calculation

Although no binary point is built into the computer, one can think of such a point existing between any two successive binary bits of a 36-bit word. The position of such a point is determined by the choice of scale factor.

Scaling a problem connotes initially defining the position of the binary point in each input number on the basis of the bounds on the number's magnitude, following the behavior of the point in all computational steps (that is, the place of the point after multiplication and division, again taking into account the bounds on the magnitude of the intermediate answers) and, finally, knowing the place of the binary point in each of the final results. Fixed-point scaling generally requires complete and accurate information about the bounds on the magnitude of all numbers that come into the computation (input, intermediate, output).

The purpose of a careful scaling analysis is to make the most efficient use of the 35 bits in the full word (minimize the presence of leading zeros and, thus, increase accuracy) and at the same time provide for the largest numbers so that no overflow (loss of the most significant bits in a number) occurs.

It is convenient to write the symbols for the numbers in a problem in a form that explicitly states the position of the binary point. We write:

$$x = 2^q \bar{x}$$

Where:

$x$  = the true value

$\bar{x}$  = a 35-bit fraction (point before the first bit), the so-called "scaled form" of  $x$ .

$2^q$  = the *scale factor*, generally such that  $q$  is the smallest integral power of 2 that makes  $2^q$  greater than the maximum value for  $x$ .

The standard convention is to consider fixed-point numbers as fractions and to express the numbers in a problem as fractions multiplied by scale factors. We are able logically to look at the 35 bits in the full word in two ways: with the binary point at the extreme left, it is  $\bar{x}$ , the scaled fraction; with the point  $q$  places from the left, it is  $x$ , the true value.

Neither adding nor subtracting changes the position of the point. However, both in multiplication and division, special consideration must be given to the place of the point in the product and in the quotient. The following rules apply for fixed-point calculation:

**Multiplication:** Two full-word fractions yield a product that is a 70-bit fraction.

Multiplicand	(35 bits)	Storage Register
Multiplier	(35 bits)	MQ Register
Product	(70 bits)	AC-MQ

**Division:** the divisor must be larger than the dividend. If this is so, a 70-bit fractional dividend divided by a 35-bit fractional divisor yields a 35-bit fractional quotient. Generally, the dividend is a 35-bit fraction and the 70-bit AC-MQ space is used to shift the dividend to the right the number of places required to make it less than the divisor.

Dividend	(70 bits)	AC-MQ
Divisor	(35 bits)	Storage Register
Quotient	(35 bits)	MQ Register
Remainder	(35 bits)	AC

It is seen that the binary point remains at the extreme left in both multiplication and division of fractions. It is for this convenience that scaling is mostly done in terms of *fractions* and scale factors.

The steps to follow in scaling are:

1. Find the bounds on the absolute values of the numbers.
2. Set up the scaling relationship between true numbers and scaled fractions by determining the required scale factor:

$$x = 2^q \bar{x}$$

3. By substitution, obtain from the "true value formula" the "scaled value formula" and write the program directly from the latter. Scale factors that do not cancel specify the required shift operations.

### Multiplication Scaling

$xy = z$     The expression  $|x|$  means the absolute value of  $x$ ; the symbol  $<$  means less than.

$$\text{Step 1: } \begin{cases} |x| < 1000 \\ |y| < 50 \\ |z| < 4000 \end{cases}$$

Frequently, additional information puts a bound on  $|z|$  which is less than the bound: max.  $x$  times max.  $y$ . In this case, we want to take advantage of the fact that  $|z|$  has an effective bound of, for example, 4,000 in place of 50,000.

$$\begin{aligned} \text{Step 2: } x &= 2^{10} \bar{x} & (2^{10} = 1024) \\ y &= 2^6 \bar{y} & (2^6 = 64) \\ z &= 2^{12} \bar{z} & (2^{12} = 4096) \end{aligned}$$

Note that in each case,  $q$  is the smallest integral power of 2 that makes  $2^q$  greater than the absolute bound. In effect, we are placing the point in the full word so that no more binary places are carried to the left of the point than are necessary to represent the absolute bound.

$$\begin{aligned} \text{Step 3: } xy &= z \\ 2^{10} \bar{x} 2^6 \bar{y} &= 2^{12} \bar{z} \end{aligned}$$

which reduces to:

$$2^4 \bar{x} \bar{y} = \bar{z}$$

Coding directly in terms of these fractions we write:

```
LDQ L( $\bar{x}$ )
MPY L( $\bar{y}$ )
LLS 4
STO L( $\bar{z}$ )
```

where  $L(\bar{x})$  means the "location of  $x$ ".

In the first two steps, we multiply two fractions to generate a 70-bit fractional product. Because the effective bound of  $|z|$  is 4,000, not 50,000, four leading binary zeros will always appear in the product and the long left shift of four eliminates these and, at the same time, brings four additional bits of accuracy from the MQ. The last step stores the fraction product,  $\bar{z}$ , in core storage.

We are able logically to look at the 35 bits in this cell in two ways: with the binary point at the extreme left it is  $\bar{z}$ , the scaled fraction; with the point 12 places over from the left, it is  $z$ , the true answer. The scaling relationship

$$z = 2^{12} \bar{z}$$

explicitly states this.

### Division Scaling

$$\frac{x}{y} = z$$

$$\text{Step 1: } \begin{cases} |x| < 1000 \\ 2 < |y| < 50 \\ |z| < 200 \end{cases}$$

The lower bound as well as the upper bound for all divisors must be known in the case where no information is available on the bound of the quotient. The bounds on  $|x|$  and  $|y|$  imply an absolute bound on  $|z|$  of 500. Frequently, as in this example, we are given a lower figure, 200, as a usable bound. We again take advantage of this in our scaling to achieve more accuracy to the right of the point.

$$\begin{aligned} \text{Step 2: } x &= 2^{10} \bar{x} & (2^{10} = 1024) \\ y &= 2^6 \bar{y} & (2^6 = 64) \\ z &= 2^8 \bar{z} & (2^8 = 256) \end{aligned}$$

Step 3:

$$\begin{aligned} \frac{x}{y} &= z \\ \frac{2^{10} \bar{x}}{2^6 \bar{y}} &= 2^8 \bar{z} \end{aligned}$$

which reduces to

$$\frac{2^{-4} \bar{x}}{\bar{y}} = \bar{z}$$

which we proceed to code as:

```
LDQ L(o)
CLA L( $\bar{x}$ )
LRS 4
DVP L( $\bar{y}$ )
STQ L( $\bar{z}$ )
```

The long right shift of four derived from the bounds on  $x$ ,  $y$ , and  $z$  prevents a divide check. Effectively, here we are dividing the fraction  $(2^{-4} \bar{x})$  by the fraction  $\bar{y}$  to obtain the quotient fraction  $\bar{z}$ .

### Accumulation Scaling

Numbers added or subtracted in the accumulator must have the same point. To prevent an overflow in the summing process, however, it is not enough to scale the final sum according to its bound but, generally (unless special logic instructions are programmed), it must be scaled by the largest bound that applies to any element in the sum or partial sum ( $P_i$ ) generated in the process of summing.

To scale the sum

$$A = \sum_{i=1}^n a_i$$

the following is usually done: If we are given the bounds

$$\begin{cases} |A| < A' \\ |a_i| < a'_i & i = 1, 2, \dots, n \\ |P_i| < P'_i & i = 1, 2, \dots, n-2 \end{cases}$$

Select the largest bound from  $A'$ ,  $a'_1$ ,  $a'_2, \dots, a'_n$ ,  $P'_1$ ,  $P'_2, \dots, P'_{n-2}$ . The largest of these is then used as the effective bound to scale both the sum  $A$  and the elements  $a_i$ .

In the more common cases, where the partial sums are not known, the bound used for selecting the scale factor is  $n |a_m|$  where  $n$  is the number of elements and  $a_m$  is the element with the greatest magnitude. In this case, the largest element is known but the remaining elements and the order in which the elements are summed do not have to be known.

## Exercises In Scaling

### 1. Scale and code:

$$A = a_1 + a_2 + a_3 + a_4$$

$$\text{given that } \begin{cases} |a_1| < 200 \\ |a_2| < 300 \\ |a_3| < 600 \\ |a_4| < 700 \\ |A| < 1000 \end{cases}$$

$$|\text{partial sums}| < 1000$$

$$\text{Solution: } a_1 = 2^{10} \bar{a}_1 \quad (2^{10} = 1024)$$

$$a_2 = 2^{10} \bar{a}_2$$

$$a_3 = 2^{10} \bar{a}_3$$

$$a_4 = 2^{10} \bar{a}_4$$

$$A = 2^{10} \bar{A}$$

$$\bar{A} = \bar{a}_1 + \bar{a}_2 + \bar{a}_3 + \bar{a}_4$$

Code:

```
CLA L(\bar{a}_1)
ADD L(\bar{a}_2)
ADD L(\bar{a}_3)
ADD L(\bar{a}_4)
STO L(\bar{A})
```

If the bound on the partial sums had not been known, 1,800 would have been used to select the scale factor, which would then have been  $2^{11} = 2,048$ . If the bound on only the largest element had been known, then 2,800 would have been used to select the scale factor, which would then have been  $2^{12} = 4,096$ . (In this case  $n = 4$  and  $|a_m| < 700$ ).

### 2. Scale and Code:

$$A = \frac{a_1 a_2 a_3}{a_4 a_5}$$

$$\text{given that } \begin{cases} |a_1| < 200 \\ |a_2| < 300 \\ |a_3| < 600 \\ 200 < |a_4| < 700 \\ 1000 < |a_5| < 1500 \end{cases}$$

Solution: The implied maximum bound for A is:

$$|A| < 180$$

$$a_1 = 2^8 \bar{a}_1$$

$$a_2 = 2^9 \bar{a}_2$$

$$a_3 = 2^{10} \bar{a}_3$$

$$a_4 = 2^{10} \bar{a}_4$$

$$a_5 = 2^{11} \bar{a}_5$$

$$A = 2^8 \bar{A}$$

Substituting:

$$2^8 \bar{A} = \frac{2^8 \bar{a}_1 \cdot 2^9 \bar{a}_2 \cdot 2^{10} \bar{a}_3}{2^{10} \bar{a}_4 2^{11} \bar{a}_5}$$

Solved for A:

$$\bar{A} = \frac{2^{-2} (\bar{a}_1 \bar{a}_2 \bar{a}_3)}{\bar{a}_4 \bar{a}_5}$$

We code:

```
LDQ L(\bar{a}_4)
MPY L(\bar{a}_5)
STO L(\bar{a}_4 \bar{a}_5)
LDQ L(\bar{a}_1)
MPY L(\bar{a}_2)
LRS 35
MPY L(\bar{a}_3)
```

```
LRS 2
DVP L(\bar{a}_4 \bar{a}_5)
STQ L(\bar{A})
```

Comment: The accuracy of the computation could be improved if more information were given. The person submitting the formula should be asked for bounds for the intermediate quantities in the computation ( $a_4 a_5$ ), ( $a_1 a_2$ ), ( $a_1 a_2$ )  $a_3$ , which may be smaller than the implied maximum bounds. For example, if we were given the effective bounds:

$$|a_4 a_5| < 750,000$$

$$|a_1 a_2| < 32,000$$

$$\text{then } |A| < 96$$

We could then scale:

$$a_4 a_5 = 2^{20} (\bar{a}_4 \bar{a}_5)$$

$$a_1 a_2 = 2^{15} (\bar{a}_1 \bar{a}_2)$$

$$A = 2^7 \bar{A}$$

and have

$$2^7 \bar{A} = \frac{2^{15} (\bar{a}_1 \bar{a}_2) 2^{10} \bar{a}_3}{2^{20} (\bar{a}_4 \bar{a}_5)}$$

reducing to:

$$\bar{A} = \frac{2^{-2} (\bar{a}_1 \bar{a}_2) \bar{a}_3}{(\bar{a}_4 \bar{a}_5)}$$

where

$$(\bar{a}_1 \bar{a}_2) = 2^2 \bar{a}_1 \bar{a}_2$$

$$(\bar{a}_4 \bar{a}_5) = 2 \bar{a}_4 \bar{a}_5$$

The coding would then be:

```
LDQ L(\bar{a}_4)
MPY L(\bar{a}_5)
LLS 1
STO L(\bar{a}_4 \bar{a}_5)
LDQ L(\bar{a}_1)
MPY L(\bar{a}_2)
LRS 33
MPY L(\bar{a}_3)
LRS 2
DVP L(\bar{a}_4 \bar{a}_5)
STQ L(\bar{A})
```

Note that the third instruction eliminates a leading zero and the seventh instruction eliminates two leading zeros to retain more accuracy.

The programmer, in setting up the order of computation steps, should inquire about the bounds on all intermediate quantities to see if it is necessary to use the implied maximum bound or whether a smaller bound may be used. Whenever there are alternatives to the order of computation steps, it is wise to choose the order that makes the most use of known effective bounds replacing implied maximum bounds.

### 3. Scale and Code:

$$P(x) = a + bx + cx^2$$

given the bounds

$$\begin{cases} |a| < 2 \\ |b| < 2 \\ |c| < 2 \\ |x| < 4 \\ |P(x)| < 30 \end{cases}$$

Solution: For convenience and economy the polynomial is factored into the form:

$$P(x) = a + x(b + cx)$$

The scale for  $|P(x)|$  is  $2^5$ . Thus

$$P(x) = 2^5 P(x) = a + x(b + cx)$$

$$P(x) = \frac{a}{2^5} + \frac{x}{2^5} (b + cx)$$

Even though this assures that the result  $\overline{P(x)}$  will be less than 1, we must now concern ourselves with scaling all intermediate results to less than 1.

$$\text{Since } |A| < 2, \text{ then } \frac{|a|}{2^5} < \frac{1}{16}$$

$$\text{Since } |x| < 4, \text{ then } \frac{|x|}{2^2} < 1$$

Therefore we write

$$\overline{P(x)} = \frac{a}{2^5} + \frac{x}{2^2} \left( \frac{b}{2^3} + \frac{cx}{2^3} \right)$$

Now we would like to scale  $\frac{|b|}{2^3} + \frac{|c||x|}{2^3}$  to  $< \frac{15}{16}$

$$\text{Since } |b| < 2 \text{ then } \frac{|b|}{2^3} < \frac{1}{2^2}$$

$$\text{Since } |c| < 2 \text{ and } |x| < 4 \text{ then } \frac{|c||x|}{2^3} < 1$$

Therefore  $\frac{|b|}{2^3} + \frac{|c||x|}{2^3} < \frac{5}{4}$

This can be converted to  $< \frac{15}{16}$  by applying an additional scale of 2.

Thus we have

$$\overline{P(x)} = \frac{a}{2^5} + \frac{x}{2^2} \frac{b}{2^3} + \frac{cx}{2^3} = 2 \left[ \frac{a}{2^6} + \frac{x}{2^2} \left( \frac{b}{2^4} + \frac{cx}{2^4} \right) \right]$$

Hence the following scaling:

$$a = 2^6 \bar{a}$$

$$x = 2^2 \bar{x}$$

$$b = 2^4 \bar{b}$$

$$c = 2^2 \bar{c}$$

$$\overline{P(x)} = 2[\bar{a} + \bar{x}(\bar{b} + \bar{c}\bar{x})]$$

LDQ L( $\bar{x}$ )

MPY L( $\bar{c}$ )

ADD L( $\bar{b}$ )

LRS 35

MPY L( $\bar{x}$ )

ADD L( $\bar{a}$ )

ALS 1

STO L [ $\overline{P(x)}$ ]

## Summary

1. With integer times integer and the result less than  $2^{35}$ , the answer is in the MQ register. Thus:

```
LDQ A
MPY B
STQ C (answer)
```

2. With fraction ( $B = 0$ ) times integer, the integer part is in the AC with the fractional part in the MQ. Thus:

```
LDQ A
MPY B      Or, rounded
STO INT    MPY B
STQ FRAC   LLS 1
           ACL ONE
           LRS 1
           STO RINT (rounded
                    integer)
```

3. With fraction ( $B = 0$ ) times fraction ( $B = 0$ ), the significant part is in the AC. Thus:

```
LDQ A
MPY B
STO FRAC (significant part), or could be rounded as in 2.
```

4. Integer divided by integer equals an integer quotient and integer remainder. Thus:

```
LDQ DVDND
PXD 0,0    Clear AC
LLS 0      Set sign of AC
DVP DVSOR
STQ QUOT   Integer quotient if DVSOR ≠ 0
STO REM    Integer remainder if DVSOR ≠ 0
```

5. Integer divided by integer may equal a fractional quotient and a trivial remainder. Thus:

```
CLA DVDND
LDQ ZERO   Clear MQ
DVP DVSOR
STQ FRAC   Fractional quotient if DVSOR > DVDND
```

Observe that the scaling of any problem is not necessarily unique. There may be several good approaches to scaling any one problem. The twin goals should always be to obtain scaling that is workable and that preserves the greatest amount of accuracy.



## Appendix G. Problem Answers

The problem answers presented do not use the most sophisticated programming concepts but simply present one way of solving the particular problem using only instructions and concepts described in text up to the point where the problem is presented.

Problem answers are identified by problem number and the page number on which the problems appear. In most cases MAP language is used, but the ORG END, etc. pseudo-operations are not included in the problem solutions.

### Problem 1, page 21

$$\begin{array}{r}
 8 \overline{)89} \\
 8 \overline{)11} \\
 8 \overline{)1}
 \end{array}
 \begin{array}{l}
 \text{Remainder} \\
 \\
 \\
 \end{array}
 \begin{array}{l}
 \uparrow 1 \\
 3 \\
 1
 \end{array}
 \text{ Answer} = 131$$

### Problem 2, page 21

	010	001	110	010	111
$\times 2$	$\overline{)0}$				
	$\overline{)1}$				
	$\overline{)2}$				
$\times 2$	$\overline{)4}$				
	$\overline{)0}$				
	$\overline{)4}$				
$\times 2$	$\overline{)8}$				
	$\overline{)0}$				
	$\overline{)8}$				
$\times 2$	$\overline{)16}$				
	$\overline{)1}$				
	$\overline{)17}$				
$\times 2$	$\overline{)34}$				
	$\overline{)1}$				
	$\overline{)35}$				
$\times 2$	$\overline{)70}$				
	$\overline{)1}$				
	$\overline{)71}$				
$\times 2$	$\overline{)142}$				
	$\overline{)0}$				
	$\overline{)142}$				
$\times 2$	$\overline{)284}$				
	$\overline{)0}$				
	$\overline{)284}$				
$\times 2$	$\overline{)568}$				
	$\overline{)1}$				
	$\overline{)569}$				

					569
					$\times 2$
					$\overline{)1138}$
					$\overline{)0}$
					1138
					$\times 2$
					$\overline{)2276}$
					$\overline{)1}$
					2277
					$\times 2$
					$\overline{)4554}$
					$\overline{)1}$
					4555
					$\times 2$
					$\overline{)9110}$
					$\overline{)1}$
					9111

Or

$$010 \ 001 \ 110 \ 010 \ 111_2 = 21627_8$$

	$\times 8$	$\overline{)2}$	$\overline{)1}$	$\overline{)6}$	$\overline{)2}$	$\overline{)7}$	
	$\overline{)16}$						
	$\overline{)1}$						
	$\overline{)17}$						
	$\times 8$						
	$\overline{)136}$						
	$\overline{)6}$						
	$\overline{)142}$						
	$\times 8$						
	$\overline{)1136}$						
	$\overline{)2}$						
	$\overline{)1138}$						
	$\times 8$						
	$\overline{)9104}$						
	$\overline{)7}$						
	$\overline{)9111}$						

### Problem 3, page 21

	$\times 8$	$\overline{).}$	$\overline{)3}$	$\overline{)5}$	$\overline{)8}$		
	$\overline{)2}$						
	$\overline{)864}$						
	$\times 8$						
	$\overline{)6}$						
	$\overline{)912}$						
	$\times 8$						
	$\overline{)7}$						
	$\overline{)296}$						
	$\times 8$						
	$\overline{)2}$						
	$\overline{)2368}$						

= .2672

### Problem 4, page 21

	$\times 8$	$\overline{)139.247}$	$\overline{)3}$	$\overline{)1}$	$\overline{)2}$		
	$\overline{)8}$						
	$\overline{)1139}$						
	$\overline{)8}$						
	$\overline{)17}$						
	$\overline{)8}$						
	$\overline{)2}$						

= 213.

Answer = 213.1763

	$\times 8$	$\overline{).}$	$\overline{)247}$				
	$\overline{)1}$						
	$\overline{)976}$						
	$\times 8$						
	$\overline{)7}$						
	$\overline{)808}$						
	$\times 8$						
	$\overline{)6}$						
	$\overline{)464}$						
	$\times 8$						
	$\overline{)3}$						
	$\overline{)712}$						

= .1763

### Problem 5, page 21

	$\overline{)18}$						
	$\overline{)2}$						
	$\overline{)9}$						
	$\overline{)2}$						
	$\overline{)4}$						
	$\overline{)2}$						
	$\overline{)2}$						
	$\overline{)1}$						

= 10010

	$\overline{)92}$						
	$\overline{)2}$						
	$\overline{)46}$						
	$\overline{)2}$						
	$\overline{)23}$						
	$\overline{)2}$						
	$\overline{)11}$						
	$\overline{)2}$						
	$\overline{)5}$						
	$\overline{)2}$						
	$\overline{)2}$						
	$\overline{)1}$						

= 1011100

10010  
1011100  
Answer = 1101110

Problem 6, page 21

$$\begin{array}{r} 2 \overline{)34} \quad 0 \\ 2 \overline{)17} \quad 0 \\ 2 \overline{)8} \quad 1 \\ 2 \overline{)4} \quad 0 \\ 2 \overline{)2} \quad 0 \\ \quad 1 \quad 0 \\ \quad \quad 1 \end{array} = 100010$$
  

$$\begin{array}{r} 2 \overline{)71} \quad 1 \\ 2 \overline{)35} \quad 1 \\ 2 \overline{)17} \quad 1 \\ 2 \overline{)8} \quad 1 \\ 2 \overline{)4} \quad 0 \\ 2 \overline{)2} \quad 0 \\ \quad 1 \quad 0 \\ \quad \quad 1 \end{array} = 1000111$$

Complement 100010      1011101  
 Add 1000111          1000111  
 Result is              0100100  
                                     1  
 Answer = 0100101

Problem 7, page 21

$$\begin{array}{r} 2 \overline{)17} \quad 1 \\ 2 \overline{)8} \quad 0 \\ 2 \overline{)4} \quad 0 \\ 2 \overline{)2} \quad 0 \\ \quad 1 \quad 0 \\ \quad \quad 1 \end{array} = 10001$$
  

$$\begin{array}{r} 2 \overline{)43} \quad 1 \\ 2 \overline{)21} \quad 1 \\ 2 \overline{)10} \quad 1 \\ 2 \overline{)5} \quad 0 \\ 2 \overline{)2} \quad 1 \\ \quad 1 \quad 0 \\ \quad \quad 1 \end{array} = 101011$$

	AC	MQ	CTR	Y
Start	000000	101011	6	010001
Add Y	010001	101011		
Shift	001000	110101	5	
Add Y	011001			
Shift	001100	111010	4	
Shift	000110	011101	3	
Add Y	010111			
Shift	001011	101110	2	
Shift	000101	110111	1	
Add Y	010110			
Shift	001011	011011	0	

Answer = 00101101101<sub>2</sub> or 1333<sub>8</sub> or  

$$\begin{array}{r} \times 8 \\ 8 \\ +3 \\ \hline 11 \\ \times 8 \\ 88 \\ +3 \\ \hline 91 \\ 43 \\ \times 17 \\ 301 \\ 43 \\ \hline 731_{10} \text{ Or } 731_{10} \end{array}$$
 Answer proof

Problem 8, page 21

$448_{10} \div 14_{10}$ . Using the conversion charts:  
 $448_{10} = 700_8 = 111\ 000\ 000_2$   
 $14_{10} = 16_8 = 000\ 001\ 110_2$

	AC	MQ	CTR	Y
Start	000111	000000	6	001110
Shift	001110	000000	5	
Sub; 1 to MQ 35	000000	000001		
Shift	000000	000010	4	
Shift	000000	000100	3	
Shift	000000	001000	2	
Shift	000000	010000	1	
Shift	000000	100000	0	

$$\begin{array}{r} 32 \\ 14 \overline{)448} \\ \underline{42} \\ 28 \\ \underline{28} \\ 0 \end{array}$$

Answer = 000 000 100 000<sub>2</sub> or 40<sub>8</sub> or 32<sub>10</sub>

Problem 9, page 21

	0	1	10	11	100
0	0	1	10	11	100
1	1	10	11	100	101
10	10	11	100	101	110
11	11	100	101	110	111
100	100	101	110	111	1000

Add Table

	0	1	10	11	100
0	0	0	0	0	0
1	0	1	10	11	100
10	0	10	100	110	1000
11	0	11	110	1001	1100
100	0	100	1000	1100	10000

Multiply Table

Problem 10, page 45

Location	Operation	Address, Tag, Decrement/Count
1 2 6 7 8	CLA	LOCA
	ADD	LOCB
	SUB	LOCC
	ADD	LOCD
	STO	LOCE

Problem 11, page 45

Location	Operation	Address, Tag, Decrement/Count
1 2 6 7 8	CLA	LOCD
	SUB	LOCE
	STO	TEMP
	CLA	LOCA
	SUB	LOCB
	ADD	LOCC
	SUB	TEMP
	STO	LOCF

Problem 12, page 47

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		CLA	PYRCD+1
		ADD	PYRCD+2
		SUB	PYRCD+3
		STO	PYRCD+4

Problem 13, page 48

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		CLA	PRTIN
		ADD	PRTIN+1
		ADD	PRTIN+2
		STO	SKBAL
		ADD	PRTIN+3
		SUB	PRTIN+4
		STO	AVAIL

Problem 14, page 49

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		LDQ	LOCA
		MPY	LOCX
		STQ	TEAX
		CLA	TEAX
		ADD	LOCB
		STO	TAXB
		LDQ	TAXB
		MPY	LOCX
		MPY	LOCX
		STQ	TAXB X
		LDQ	LOCX
		MPY	LOCC
		STQ	TEMCX
		CLA	TEMCX
		DVP	LOCD
		STQ	TCXD
		CLA	TAXB X
		ADD	TCXD
		STO	ANS

Problem 15, page 49

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		CLA	LOCX1
		DVP	LOCX4
		MPY	LOCX2
		DVP	LOCX5
		MPY	LOCX3
		STO	ANS

Problem 16, page 52

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		CLA	LOCA
		ADD	LOCB
		ADD	LOCC
		ALS	1
		STO	LOCP

Problem 17, page 52

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		CLA	LOCA
		ADD	LOCB
		SUB	LOCC
		ARS	4
		STO	LOCP

Problem 18, page 52

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		CLA	LOCA
		SUB	LOCB
		ALS	3
		STO	LOCP

Problem 19, page 54

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		CLA	LOCA
		CAS	LOCB
		CLA	LOCB
		TRA	ONE
ONE		CAS	LOCC
		CLA	LOCC
		TRA	TWO
TWO		STO	LOW

Problem 20, page 54

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		CLA	FIND
		CAS	PARTN
		TRA	QUIT
		TRA	PROCS
		TRA	MORE

Problem 21, page 54

Location				Operation	Address, Tag, Decrement/Count
1	2	6	7	8	
PARTA				CLA	LOCA
				ADD	LOCB
				ADD	LOCC
				ADD	LOCD
				TOV	OFLW
PARTB				LBT	
				TRA	SHIFT
PARTC				PBT	
				TRA	END
				ARS	1
				LDQ	ONE
				STQ	PBIT
END				HPR	
OFLW				LRS	35
				STO	TEMP
				ADD	OYFLW
				STO	OYFLW
				CLA	TEMP
				LLS	35
				TRA	PARTB
SHIFT				ARS	1
				TRA	PARTB
ONE				OCT	1

Problem 21 (Cont'd)

Location				Operation	Address, Tag, Decrement/Count
1	2	6	7	8	
STOR2				ARS	35
				STO	TEMP
				CLA	TWO
				TRA	STOR1+3
STOR3				ARS	35
				STO	TEMP
				CLA	THREE
				TRA	STOR1+3
ONE				OCT	1
TWO				OCT	2
THREE				OCT	3

Problem 21 (Cont'd)

Location				Operation	Address, Tag, Decrement/Count
1	2	6	7	8	
PARTA				CLA	LOCA
				ADD	LOCB
				TOV	OYFL1
				ADD	LOCC
				TOV	OYFL2
				ADD	LOCD
				TOV	STOR1
PARTB				Same as previous example	
OYFL1				ADD	LOCC
				TOV	OYFL3
				ADD	LOCD
				TOV	STOR2
STOR1				ARS	35
				STO	TEMP
				CLA	ONE
				ADD	OYFLW
				STO	OYFLW
				CLA	TEMP
				ALS	35
				TRA	PARTB
OYFL2				ADD	LOCD
				TOV	STOR2
				TRA	STOR1
OYFL3				ADD	LOCD
				TOV	STOR3

Problem 22, page 60

Location				Operation	Address, Tag, Decrement/Count
1	2	6	7	8	
				AXT	24,1
				PXD	0,0
				MIT	N+24,1
				ADD	N+24,1
				TIX	*-2,1,1 (Note: the * means
				STO	PSUM the location of this
					instruction)

Problem 23, page 60

Location				Operation	Address, Tag, Decrement/Count
1	2	6	7	8	
				AXT	31,1
				PXD	0,0
				MIT	M+31,1
				ADD	M+31,1
				TIX	*-2,1,1 (Note: the * means
				STO	PSUM the location of this
				AXT	30,1 instruction)
				CLA	M
				ADD	M+31,1
				TIX	*-1,1,1
				SUB	PSUM
				STO	MSUM

Problem 24, page 61

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		A X T	99,1
		C L A	HUND
HUND		C A S	HUND+100,1
		T I X	*-1,1,1
		T R A	*+3
		C L A	HUND+100,1
		T I X	*-4,1,1
		S T O	PLUS
		A X T	99,1
		C L A	HUND
		C A S	HUND+100,1
		C L A	HUND+100,1
		T I X	*-2,1,1
		T I X	*-3,1,1
		S S P	
		C A S	PLUS
		C H S	
		T R A	*+2
		C L A	PLUS
		A X T	100,1
		C A S	HUND+100,1
		T I X	*-2,1,1
		T R A	EQUAL
		T I X	*-3,1,1
EQUAL		P X A	,1

Problem 24 (Cont'd)

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		C H S	
		A D D	HUND
		S T A	LRG ST

Problem 25, page 61

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		A X T	50,1
		A X T	0,2
		P X D	0,0
		C A S	DATA+50,1
		T R A	*+3
		T R A	EQUAL
		T X I	*+1,2,1
LOOP		T I X	*-4,1,1
		S X D	ANS,2
		A X T	DATA+50,1
CHNG		T I X	*+1,1,0
		S X A	ANS,1
EQUAL		S X D	CHNG,1
		T R A	LOOP

Problem 26, page 61

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		A X T	10,1
		A X T	0,2
RETN		C L A	B1
		C A S	A+10,1
		T R A	ADD1
		T I X	*-2,1,1
RETN1		T I X	*-3,1,1
		S X A	CONDA,2
TEN		A X T	10,1
		A X T	0,2
		C L A	B2
RETN2		C A S	A+10,1
		T R A	LOOP
		T R A	*+1
		T X I	*+1,2,1
LOOP		T I X	*-4,1,1
		S X A	CONDC,2
		C L A	TEN
		S U B	CONDA
		S U B	CONDC
		S T A	CONDB
ADD1		T X I	RETN,2,1

Problem 27, page 61

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
START		A X T	99,1
		A X T	0,2
		C L A	NUM+99,1
		C A S	NUM+100,1
		T R A	EXCNG
		T I X	*-3,1,1
		T I X	*-4,1,1
TEST		P X D	
		P X A	0,2
		T N Z	START
			At this point, the program is finished.
EXCNG		L D Q	NUM+100,1
		S T O	NUM+100,1
		S T Q	NUM+99,1
		T X I	*+1,2,1
		T I X	START+2,1,1
		T R A	START
		E N D	

Problem 28, page 61

Location				Operation	Address, Tag, Decrement/Count
1	2	6	7 8		
				STZ	SUM
				AXI	3, 1
LOOP				CLA	X1+3, 1
				SUB	Y1+3, 1
				STQ	SAVE
				LDQ	SAVE
				MPY	SAVE
				L.L.S	35
				ADD	SUM
				STQ	SUM
				TIX	LOOP, 1, 1
				HPR	
SAVE				BSS	1
SUM				BSS	1
X1				BSS	3
Y1				BSS	3
				END	

Problem 29 (Cont'd)

Location				Operation	Address, Tag, Decrement/Count
1	2	6	7 8		
				STQ	Y+1
				AXI	4+1
				LDQ	ONE
				CLA	X+5, 1
				CAS	Y+5, 1
				RQL	1
				TRA	*+1
				STQ	TEST+4, 1
				TIX	*-6, 1, 1
ZERO				OCT	0
ONE				OCT	1

Problem 29, page 66

Location				Operation	Address, Tag, Decrement/Count
1	2	6	7 8		
				LDQ	ZERO
				CLA	X
				LGR	7
				STQ	X+4
				LDQ	ZERO
				LGR	11
				STQ	X+3
				LDQ	ZERO
				LGR	7
				STQ	X+2
				LDQ	ZERO
				LGR	11
				STQ	X+1
				LDQ	ZERO
				CLA	Y
				LGR	7
				STQ	Y+4
				LDQ	ZERO
				LGR	11
				STQ	Y+3
				LDQ	ZERO
				LGR	7
				STQ	Y+2
				LDQ	ZERO
				LGR	11

Problem 30, page 67

Location				Operation	Address, Tag, Decrement/Count
1	2	6	7 8		
				AXI	9000, 1
ENTER				CLA	JCON
				CCS	INVPR+9000, 1, S
				TRA	R
				TRA	J
				CLA	CCON
				CCS	INVPR+9000, 1, S
				TRA	F
				TRA	C
A				CLA	AADRS
COMP				STA	TOTAL
				STA	TOTAL+1
				LDQ	INVPR+9001, 1
				MPY	INVPR+9002, 1
				L.L.S	35
TOTAL				ADD	ATOTL
				STO	ATOTL
				TIX	ENTER, 1, 3
				HPR	*
C				CLA	CADRS
				TRA	COMP
F				CLA	FADRS
				TRA	COMP
J				CLA	JADRS

Problem 30 (Cont'd)

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		TRA	COMP
R		CLA	RADRS
		TRA	COMP
CCON		BCI	1,66666C
JCON		BCI	1,66666J
AADRS		PZE	ATOTL
CADRS		PZE	CTOTL
EADRS		PZE	JTOTL
JADRS		PZE	RTOTL
RADRS		PZE	RTOTL
		END	

Problem 33, page 75

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		ENTER	AC
		STO	MQ
		STO	MQ
		CLA	0
		LRS	25
		LBT	
		TRA	HALT
		LLS	2
		LBT	
		TRA	ZERO1
		LLS	3
		LBT	
		TRA	RESTR
		LRS	2
		LBT	
		TRA	ZERO2
HALT		CLA	AC
		LDQ	MQ
		HPR	*
ZERO1		LLS	2
		LBT	
		TRA	RESTR
		LRS	1
		LBT	
		TRA	BLANK
		TRA	HALT

Problem 31, page 68

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		AXT	4,1
		CLA	CON
		TMT	25
		ADD	THRT9
		TIX	*-2,1,1
CON		PZE	69,1000
THRT9		PZE	39

Problem 33 (Cont'd)

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		ZERO2	LDQ
			MQ
			LLS
			35
			CLA
			AC
			TRA*
			0
		BLANK	CLA
			AC
			ARS
			35
			LDQ
			MQ
			TRA*
			0
		RESTR	CLA
			AC
			LDQ
			MQ
			TRA*
			0
		CONS	TRA
			ENTER
		AC	BSS
			1
		MQ	BSS
			1
			END

Problem 32, page 68

Location		Operation	Address, Tag, Decrement/Count
1	2	6 7 8	
		CLA	CON1
		TMT	45
		CLA	CON2
		TMT	30
		CLA	CON3
		TMT	15
CON1		PZE	ANSWR,,BLK1
CON2		PZE	ANSWR+45,,BLK2
CON3		PZE	ANSWR+75,,BLK3

Problem 34, page 97

Location		Operation	Address, Tag, Decrement/Count
1	2	4 7 8	
		AXI	10,1
READ		RDS	B3
		RCHB	10COM
		TCOB	*
		TRCB	REDUN
		TEFB	EOF
		HPR	1
REDUN		B.S.R	B3
		TLX	READ,1,1
		HPR	4681
EOF		HPR	32767
10COM		PTH	INARA, ,15
INARA		B.S.S	15
B3		EQV	1171



Add	44	Data Track Address	109
Add Tables	14	Data Transmission	67
Adders	26	Data Transmission Unit, IBM 1009	82
Adding BCD Coded Numbers	66	Decimal Data	15, 40
Addition Overflow	25	Decimal Integer	39
Address Counter	86	Decimal to Octal	18
Address Field	23	Decimal to Binary	19
Address Modification	11	Decrement Field	54
Arithmetic Tables	14	Delayed Traps	116
Assembly Process	28	Destructive Address Modification	55
Assembly Programs	30	Direct Data Operation	112
Assembly Register	86	Direct Data Connection	113
Base Elements of Programming System	30	Direct Data Trap	118
Basic Monitor	36	Disk Gaps	109
Begin	42	Disk Operation	109
Binary Addition	16	Disk Programming Examples	110
Binary Coded Information	41	Disk Storage iocs	35
Binary Indicators	15	Disk Storage, IBM 1301	82
Binary Mode	15	Divide	46
Binary Multiplication	17	Divide Example	49
Binary Number System	15	Double-Precision Format	74
Binary Place Part	40	E Cycle Flow Lines	27
Binary Representations	16	Edit Source Program	32
Binary Subtraction	17	Effective Address	56
Binary System	88	Elements	39
Binary to Decimal	19	End	42
Binary to Octal	18	End-of-File Indicator	95
Block Diagram	8	End-of-File Procedures	35
Block Ending with Symbol	41	End-of-File Gaps	88
Block Starting with Symbol	41	End-of-Reel Procedures	35
Blocking and Deblocking	34	Equal	41
Boolean	42	Error Correction Procedures	35
Buffers	77	Execute Cycle	27
Calculating	7	Execution Process	29
Card Read Punch, IBM 1402	81, 101	Exponent Part	39, 40
Card Read Punch, IBM 1622	81, 100	Exponents	70
Card Reader Operation	101	Expression	39
Card Stackers	100	Extra Channel Traps	127
Carriage Control	106	Fixed and Variable Word Length	6
Channel Data Register	86	Fixed-Point Number	40
Channel-in-Use Indicator	91	Floating-Point Format	69
Channel Trap Stores	119	Floating-Point Conversions	69
Channel Traps	77	Floating-Point Examples	70
Check Sums	64	Floating-Point Number	39
Code Checking	20	Floating-Point Summary	71
Code Combinations	22	Floating-Point Trap	74
Collating Sequence	23	Format Track	109
Column Binary	99	Fractions	19
Comments Field	38, 44	Generator Programs	33
Comparing	10	Halt and Proceed	116
Compilers	31	Home Address	109
Complement Arithmetic	56	I Cycle Flow Lines	26
Computing Addresses	63	iocs Level Concept	77
Conditional Transfer	8	iocs Summary	78
Control Cards	37	IBJOB Processor Flow	37
Count Control	9	IBSYS Input	38
Counters	26	IBSYS System Components	36
Cylinder Concept	108	Improper Fractions	20
Data Channel Reset	96	Improper Length Record Procedures	35
Data Channel Store	96	Inclusive OR	57, 65
Data Channel A Operation	84	Index Data Flow	61
Data Channel (IBM 7904) Operation	86	Index Point	109
Data Generating Operations	40		

Index Register Arithmetic	57	Principal Part	39, 40
Index Servicing	58	Print Operation	105
Index Transfer Instructions	60	Printer, IBM 1403	81
Indexing	12	Processing	5
Indexing Concept	56	Processing Unit Data Flow	27
Indexing Loop	12	Processor	29
Indirect Address	12, 22, 55, 62	Program Indicators	89
Input	5	Program Loop	8
Input Data	5	Program Switch	11
Input/Output Check Indicator	96	Pseudo Operation	38
Input/Output Control Systems	33	Random and Sequential Processing	78
Input/Output Scheduling	34	Read Operation	85, 101, 102
Instruction Cycle	26	Reading Data	6
Instruction Format	5	Ready Test	105
Instruction Modification	10	Record Address	109
Instructions	5	Redundancy Check Indicator	92
Instructions and Data	23	Register Shifting	25
Integers	18	Registers	24
Inter-Record Gaps	88	Release Protect Mode	118
Interval Timer Reset	116	Remote Inquiry Unit, IBM 1014	82
Interval Timer Overflow	118	Rewind Instructions	96
Labels	76	Scale	7
Literals	39	Select Instructions	104
Load and Go	29	Sequence Checking	10
Loader	36	Serial and Parallel Operation	6
Location Counter Operations	42	Seven-Bit Code	20, 87
Location Field	38, 43	Single-Character Operation	106
Logic Operations	8	Source Program	28
Logical Check Sums	63	Special Print Conditions	106
Logical or	57	Special Read Conditions	101, 102
Machine Cycles	26	Special Write Conditions	102, 103
Machine Language	33	Spill	74
Macro Assembly Program Language	39	Stacker Select Example	104
Macro Generator Use	33	Standard Error Correction Routines	34
Macro Generators	32	Storage Allocation Operations	41
Magnetic Disk Recording	108	Storage Parity	117
Magnetic Tape	87	Stored Program Concepts	5
Masking	65	Subroutine	9
Maximum	41	Subtract	44
Memory Protect	117	Symbol Defining Operations	41
Minimum	42	Symbolic Assembly	33
Multiple Tags	57	Symbolic Coding	43
Multiply	45	Symbolic Instructions	39
Multiply Example	47	Tag Bits	56
Multiply Tables	14	Tape Labeling	35
Normal Form	70	Tape Marks	94
Number Conversions	18	Tape Records	88
Object Program	28	Tape Unit, IBM 729	81
Octal Data	40	Tape Unit, IBM 7330	82
Octal Number System	17	Terms	39
Octal to Binary	18	Translation	29
Octal to Decimal	18	Transmit Example	68
Off Line	29	Trap Execution	127
One's Complement	56	Trap Stores	127
Operand	5	Trapping Priority	116
Operating System	36	Two's Complement	56
Operation Codes	39	Typewriter Busy	106
Operation Field	38, 43	Typewriter Operation	106
Operation Part	5	Typewriter Shifting	107
Operators	39	Un-normal Form	70
Origin	42	Unpack	65
Output	5	Unusual Condition Routines	34
Overlap and Data Channels	76	Use	42
Packed Word	64	Variable Field	38, 44
Paper Tape Reader, IBM 1011	82	Variable Length Formats	48
Parity Checking Instructions	63	Word Counter	86
Partial Store Instructions	57	Write Operation	85, 102
Place Value	16		