File Number 1440-01
Form A24-3116-0

IBM Systems Reference Library

System Operation Reference Manual
IBM 1440 Data Processing System

This publication contains the instruction set for the IBM 1440 and the formula for calculating the execution time of each instruction. The operation code for every instruction is given in actual and mnemonic form, with examples of each.

The instructions and applicable timings for the input/output printer on the IBM 1447 Console are discussed.

For general information on units attached to the 1440, refer to the IBM 1440 *Bibliography*, Form A24-3005. For instructions and applicable timings for attached units, see:

- *Miscellaneous Input/Output Instructions* (1440), Form A24-3117.
- *Tape Input/Output Instructions* (1401, 1440, 1460), Form A24-3069.
- *Disk Storage Input/Output Instructions* (1401, 1440, 1460), Form A24-3070.

Preface

This publication is a reference text for the IBM 1440 Data Processing System. It provides a detailed explanation of the instructions used by the system to manipulate data. Detailed explanations of the instructions used with the console input/output printer when it is attached to the system are also included. The reader should be familiar with the IBM *1440 Systems Summary,* Form A24-3006, and the various publications on applied programming material, such as *Autocoder.*

The manual is divided into these sections:

- Introduction
- Arithmetic Operations
- Logic Operations
- Data-Moving Operations
- Miscellaneous Operations
- Edit Operation
- IBM 1447 Console Operations

The sections are independent and do not have to be used in the order in which they appear.

The publication is intended for programmers and systems personnel who have a general knowledge of the IBM 1440 Data Processing System and who require a reference text for detailed information. It can also be used as a training aid in the instruction of programmers and operators.

It should be noted that other publications referenced here are, in most cases, prerequisites for a complete understanding of the material presented in this publication.

This publication, Form A24-3116-0, is a major revision and consolidation of the applicable material from:

A26-5666, and includes the applicable material from the following Technical Newsletter:

N24-0062

The original publication and the applicable Technical Newsletter are obsoleted by this publication.

This publication, Form A24-3116-0, also obsoletes the console I/O printer portions of:

A26-5667

Refer to IBM *1440 Bibliography,* Form A24-3005, for other publications.

Copies of this and other IBM publications can be obtained through IBM Branch Offices.
Address comments regarding the content of this publication to IBM Product Publications, Endicott, New York

Contents

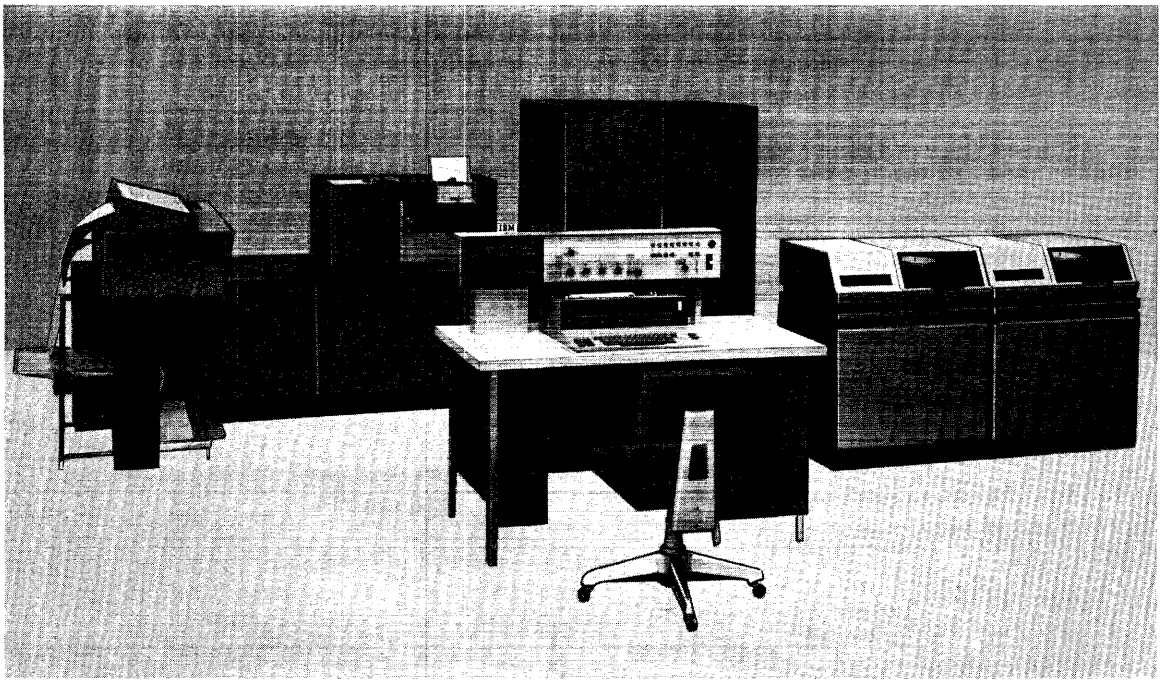


Figure 1. IBM 1440 Data Processing System

The IBM 1440 Data Processing System (Figure 1) represents a major advance in low-cost data processing systems. The IBM 1440 offers small companies the functional capabilities of large data processing systems, but at speeds and costs in keeping with their needs and abilities. The input and output devices of the 1440 enable it to be effective in system areas where there has long been a need for a data processing system but not the volume of work to justify such a system. Processing methods of the 1440 are similar to those of the IBM 1401 Data Processing System.

The IBM 1440 is a solid-state system with compact components and input/output devices that enable it to be located in an area of approximately 16′ x 22′. In addition to its features of compactness and low-cost, the 1440 presents a new concept in data processing with the introduction of the removable disk pack.

In 1953, the introduction of IBM magnetic tape systems provided data processing systems with the ability to process large volumes of input and output data at very high speeds. Magnetic tape offers the advantage of providing virtually unlimited storage capacity. In 1956, the RAMAC® disk file introduced a new concept in data processing, permitting, as it did, storage of large volumes of data that were accessible in a random sequence.

The IBM 1311 Disk Storage Drive for the IBM 1440 Data Processing System provides virtually unlimited random and sequential access storage. A disk pack containing 2,000,000 characters of information can be removed from the 1440 system and another pack put into its place in one to two minutes. This operator-removable disk pack combines the large-volume and sequential-processing advantages of tape systems with the random-access abilities of a RAMAC file.

The ease of mobility of a disk pack (the weight of the pack is less than 10 pounds) and the simplicity of its removal from the drive means that 2,000,000 characters of data can be placed in the system within seconds. Data can be organized in the disk pack in *random* or *sequential* order; regardless of how the data is located on the disk pack, it can be retrieved by the system in a random or sequential order with equal facility, depending on individual requirements. Up to five disk drives, each equipped with one disk pack, can be attached "on line" to provide 10,000,000 characters of information available at one time (equivalent to 125,000, 80-column punched cards).

The 1440 is primarily a disk-storage oriented system, providing a group of balanced input/output devices to work in conjunction with the IBM 1441 Processing Unit and with the IBM 1311 Disk Storage Drive. For operations that require extensive calculating ability and do not need disk storage, the 1440 can function as a card system.

The IBM 1440 is available in various configurations to satisfy the requirements of individual users. It can be ordered to meet the basic requirements of an accounting system, and then increased in size as data processing requirements increase. If the 1440 is expanded to its maximum size and data processing requirements continue to grow, procedures and systems developed for the IBM 1440 can be readily adapted for processing on the medium-size IBM 1401 Data Processing System. With continued expansion and growth, adaptation to larger equipment such as the IBM 1460 and 1410 Data Processing Systems can be made.

This is why we refer to the 1440 as a member of the 1400-series family.

The Stored Program

The IBM 1440 Data Processing System performs its functions by executing a series of instructions at high speed. A particular set of instructions, designed to solve a specific problem, is known as a *program*. Because the 1440 stores its instructions internally, it is called a *stored program* system.

The 1440 system normally executes instructions sequentially. The system can also skip over a particular group of instructions, or otherwise change the sequence of the program. Branch instructions are provided in the system to make it possible to alter the program and take the next instruction from another area of the stored program. This function also makes it possible to repeat an instruction, or group of instructions, as often as desired.

A series of programmed tests determines the logical path of the program. These tests are made at various points in the program to control the course of program step execution for specific conditions that can arise during processing.

Variable Word Length

Stored programming involves the concept of *words*. A 1440 word can be a single character, or a group of characters, representing a complete unit of information. Because IBM 1440 words are not limited to a specific number of storage positions — i. e., have variable word length — and because each position of core storage is addressable, each word occupies only the number of core-storage locations actually needed for the specific instruction or data field.

WORD MARKS

The use of the variable-length instruction and data format requires a method of determining the instruction and data-word length. This identification is provided by a word mark. Word marks are illustrated by underlining the characters with which they are associated.

The word mark serves several functions:

1. Indicates the beginning of an instruction.
2. Defines the size of a data word.
3. Signals the end of execution of an instruction.

The rules governing the use of word marks are:

1. Predetermined locations for word marks are assigned in planning the program. These predetermined word marks are normally expected to remain in these locations throughout the complete program. The word marks are set into storage locations by a loading routine.
2. Word marks are not moved with data during processing, except when a *load* instruction (see No. 5 below) is used.
3. For an arithmetic operation, the *B-field* must have a defining word mark, and the *A-field* must have a word mark only when it is shorter than the B-field.
4. A load instruction moves the word mark and data from the A-field to the B-field, and clears any other word marks in the designated B-field, up to the length of the A-field.
5. When moving data from one location to another, only one of the fields need have a defining word mark, because the *move* instruction implies that both fields are the same length.
6. A word mark must be associated with the high-order character (operation code) of every instruction.
7. The 4-character BRANCH UNCONDITIONAL instruction, the 7-character SET WORD MARK, and CLEAR STORAGE AND BRANCH instructions are the only instructions that can be followed by a blank without a word mark. All other instructions must be followed by a word mark.

Two operation codes are provided for setting and clearing word marks during program execution.

Stored Program Instructions

All machine functions are initiated by instructions from the 1440 stored program. Because the 1440 uses the variable-word-length concept, the length of an instruction can vary from two to eight characters, depending on the operation to be performed.

Instruction Format

Mnemonic	*Op Code*	*A- or I-address*	*B-address*	*d-character*
X	X	XXX	XXX	X

Mnemonic. This is the mnemonic operation code that is used by the *Autocoder* processor program to designate the actual machine operation code.

Op Code. This is always a single character that defines the basic operation to be performed. A word mark is always associated with the operation code position of an instruction.

A-Address. This always consists of three characters. It can identify the units position of the A-field, or it can be used to select an input/output unit (card read-punch, disk storage unit, data transmission unit, paper tape reader, printer, tape punch, etc.).

I-Address. Instructions that can cause program branches use the I-address to specify the location of the next instruction to be executed if a branch occurs.

B-Address. This is a 3-character storage address that identifies the B-field. It usually addresses the units position of the B-field, but in some operations (such as move record or input/output operations it specifies the high-order position of a record-storage area.

d-Character. The d-character is used to modify an operation code. It is a single alphabetic, numerical, or special character, positioned as the last character of an instruction.

Examples of the five combinations possible in variable-length instructions are shown in Figure 2.

Instruction Descriptions

Specific instructions have been described in a standard format:

Title. This is the description of the instruction.

NUMBER OF POSITIONS	OPERATION	INSTRUCTION FORMAT			
2	SELECT STACKER	Op code K	d-character 2		
4	BRANCH	Op code B	I-address 400		
5	BRANCH IF INDICATOR ON	Op code B	I-address 625	d-character /	
7	ADD	Op code A	A-address 072	B-address 423	
8	BRANCH IF CHARACTER EQUAL	Op code B	I-address 650	B-address 080	d-character 4

Figure 2. IBM 1440 Instruction Formats

Instruction Format. This is the format of the particular instruction described. The mnemonic operation code used in the IBM *Autocoder* is given.

Function. This is the function of the instruction.

Word Marks. This is the effect of the word marks with regard to data fields.

Timing. This is the formula to be used in calculating the timing of the instruction. Key to abbreviations used in the formulas is shown in Figure 3.

Notes. These are special notations or additional information pertaining to the operation.

Address Registers After Operation. The contents of the address registers are represented by the codes described in Figure 4.

Example. A practical application of the instruction is described and shown as a label for the 1440 *Autocoder* language. With the label is the actual machine address in parentheses. It is not necessary for the programmer to know the actual address of a label when writing the program. The processor program assigns the actual address during the program assembly.

Assembled Instruction. This is the actual machine language instruction that is assembled by the *Autocoder* processor program from the symbolic entries shown in the example.

Key to abbreviations used in formulas:	
L_A	= Length of the A field
L_B	= Length of the B Field
L_C	= Length of Multiplicand field
L_I	= Length of Instruction
L_M	= Length of Multiplier field
L_P	= Length of Product field
L_Q	= Length of Quotient field
L_R	= Length of Divisor field
L_S	= Number of significant digits in Divisor (excludes high-order zeros and blanks)
L_W	= Length of A or B field, whichever is shorter
L_X	= Number of characters to be cleared
L_Z	= Number of characters back to rightmost zero in control field
I/O	= Timing for Input or Output cycles
F_m	= Forms movement times
Σ	= Number of fields included in an operation
N_S	= Number of disk sectors
S_S	= Number of characters in disk sector

Figure 3. Timing Formula Coding

ABBREVIATION	MEANING
A	A-address of the instruction
B	B-address of the instruction
NSI	Address of the next sequential instruction
BI	Address of the next instruction if a branch occurs
L_A	The number of characters in the A-field
L_B	The number of characters in the B-field
L_W	The number of characters in the A- or B-field, whichever is smaller
Ap	The previous setting of the A-address register
Bp	The previous setting of the B-address register
dbb	The d-character and blank in the units and tens position

Figure 4. Address Registers after Operation Coding

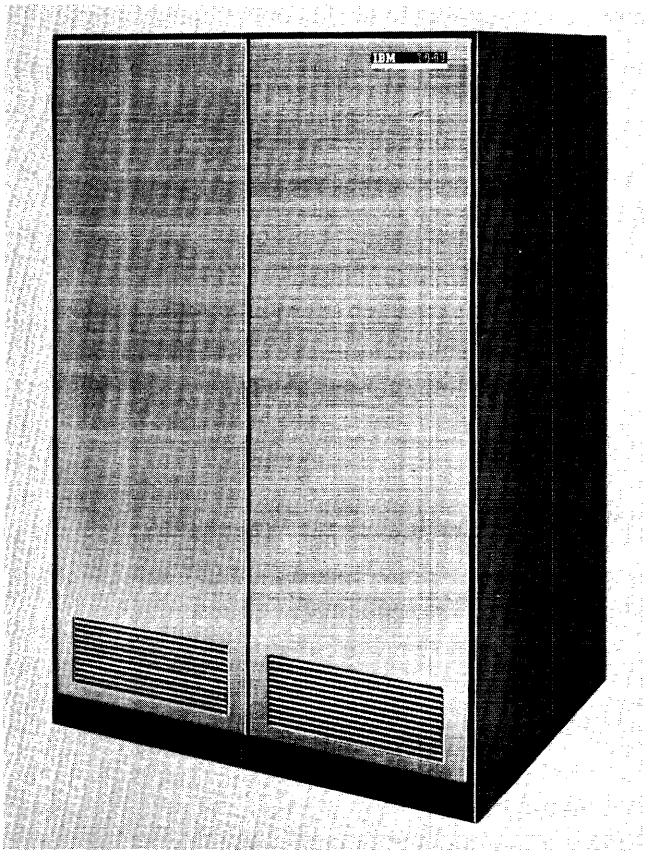


Figure 5. IBM 1441 Processing Unit

IBM 1441 Processing Unit

The IBM 1441 Processing Unit (Figure 5) is the controlling center of the IBM 1440 Data Processing System. The processing unit can be divided into two sections:

1. The arithmetic-logical section
2. The control section

The arithmetic-logical section performs such operations as addition, subtraction, transferring, comparing, and storing. By adding the multiply-divide special feature, the 1441 can perform direct multiplication and division. This section also has logical ability — the ability to test various conditions encountered during processing and to take the action called for by the result.

The control section directs and coordinates the entire system as a single multipurpose machine. These functions involve controlling the input/output units and the arithmetic-logical operation of the processing unit, and transferring data to and from storage. This section directs the system according to the procedure originated by its human operators.

Magnetic Core Storage

The IBM 1441 Processing Unit houses the magnetic-core storage area (Figure 6) that is used by the 1440 system for storing the instructions and data. The data in each core-storage position is available, in 11.1 microseconds and the design of the core-storage control circuits makes each position individually addressable. This means that an instruction can designate the exact storage locations that contain the data needed for that step.

The physical make-up of each core-storage location enables the IBM 1441 to perform arithmetic operations directly in the storage area. (This is called *add-to-storage* logic.)

Language

In the punched-card area of data processing, the language of the machine consists of holes punched in a card. As data processing needs increase, the basic card language remains the same. But in the transition from unit-record systems to the IBM 1440 Data Processing System, and from there to other computer systems, another faster, more flexible machine language emerges.

Just as each digit, letter in the alphabet, or special character is coded into a card as a punched hole or a combination of punched holes, it is coded into magnetic storage as a pattern of magnetized spots.

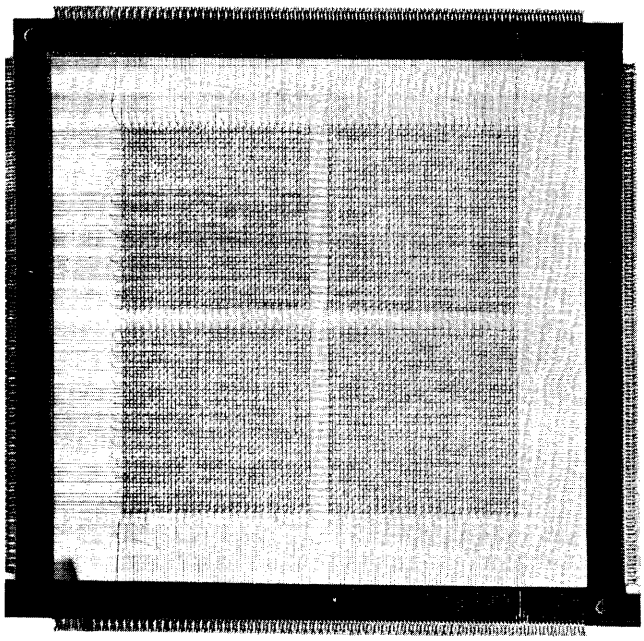


Figure 6. Magnetic Core Storage

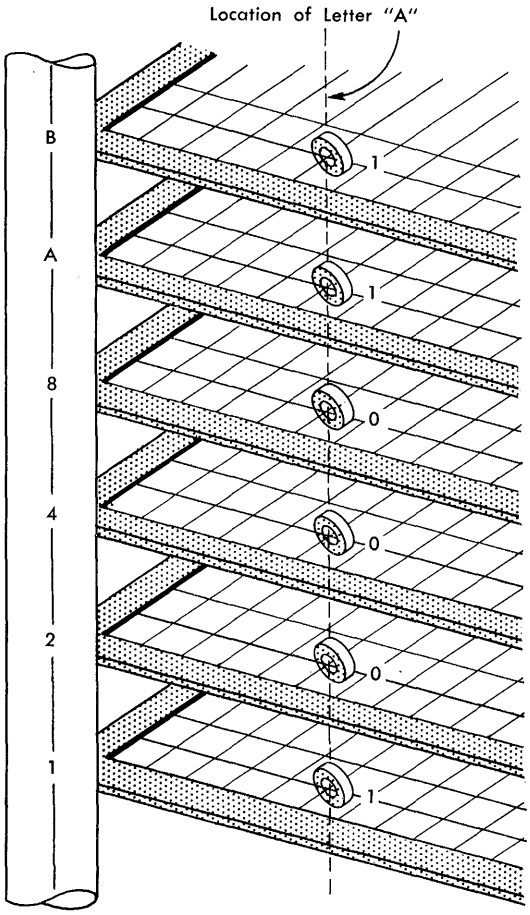


Figure 7. The Letter A Represented in Binary-Coded-Decimal Form in Core Storage

Many different code patterns can be set up. The internal code used in the IBM 1440 Data Processing System is called *binary-coded decimal* (Figure 7). All data and instructions are translated into this code as they are stored.

The numbers 0 through 9 are represented by a single bit, or a combination of bits designated 1, 2, 4, 8. Disregarding the C- or check bit, bits 2 and 8 stand for 0, bits 1 and 2 for 3, bits 1 and 4 for 5, bits 2 and 4 for 6, bits 1, and 2, and 4 for 7, and bits 1 and 8 for 9.

Letters and special characters are represented by a combinations of numerical bits (8421) and zone bits (BA). B- and A-bits, in combination, correspond to the 12-zone punch. The B-bit corresponds to the 11-zone punch, and the A-bit to the 0-zone punch. The letter C, for example, which is the third letter in the 12-zone of the alphabet (card code 12-3), is a combination of BA21 bits. BA is the same as 12, and 21 is the same as 3.

This covers six of the seven possible bits that are used to represent a character. The seventh bit (C) is a built-in checking feature that the computer automatically supplies.

Note that the check bit is not part of the character configuration when the number of BA8421 bits that represent the character is odd. It appears only for those characters where the number of bits BA8421 is even. The automatic inclusion of the check bit changes the configuration of the character from an even number of bits to an odd number of bits. Thus, all characters shown in Figure 64 are shown in the *odd-parity* mode.

Information introduced into the system is translated to the binary-coded-decimal form for use in all data flow and processing from that point on, until it is translated into printed output as reports and documents are written, or converted to punched-card code, for punched-card output. Converting input data to the 1441 internal code, and subsequently reconverting, is completely automatic.

Processing

Processing is the manipulation of data from the time it is introduced to the system as input until the desired results are ready for output. The following functions are performed in the IBM 1441 Processing Unit.

Logic

The logic function of any kind of data processing system is the ability to execute program steps; but even more, it is the ability to evaluate conditions and select alternate program steps on the basis of those conditions.

In unit-record equipment, an example of this logic is selector-controlled operations based on an X-punch or No X-punch, or based on a positive or negative value, or perhaps based on a comparison of control numbers in a given card field.

Similarly, the logic functions of the 1440 system control comparisons, branching (alternate decisions similar in concept to selector-controlled procedures), move and load operations (transfer of data or instructions), and the general ability to perform a complicated set of program steps with necessary variations.

Arithmetic

The IBM 1441 Processing Unit can add, subtract, multiply, and divide. Multiplication and division can be accomplished in any 1440 system, by programmed subroutines. When the extent of the calculations might otherwise limit the operation, a special multiply-divide feature is available.

Editing

As the term implies, editing adds significance to output data by punctuating and inserting special characters and symbols. The 1440 system has the ability to perform this function, automatically, with simple program instructions.

Internal Checking

Advanced circuit design is built into the 1440 to assure accurate results. Self-checking with the system consists of *parity* and *validity* checking.

Parity Checking

The IBM 1441 checks characters at various locations in the unit for odd-bit configurations. The 6-bit, binary-coded-decimal internal language used by the 1440 also has a check bit for odd-bit checking purposes, and a word-mark bit. The check bit is added to all characters that would otherwise have an even number of bits.

Example: A character P has a binary-coded decimal equivalent of B 4 2 1. The check bit is added to give this character an odd number of bits (C B 4 2 1).

If the character has a word mark associated with it, the word mark is included in the test for odd-bit parity.

Example: If the character P has a word mark, the check bit is not added because the bit configuration is odd (WM B 4 2 1).

Whenever a parity error occurs, a console light turns on, indicating the place where the error occurred (see IBM *1447 Console,* Form A24-3031).

Validity Checking

Validity checking is performed to detect illogical bit combinations within the systems. The type of validity checks performed are:

1. The output from the adder is checked for a logical numeric code.
2. The operation register is checked so that only valid operation codes are processed.
3. The storage address register is checked to make sure the core-storage addresses are valid addresses within the core-storage address range of that particular processing unit. Depending on the core-storage size, the units and/or hundreds address positions contain zone bits that specify blocks of addresses. (Refer to *Addressing System* section for detail information.) These zone-bit combinations are checked to make sure the combinations are addressing an installed core-storage address. A check is made to see if the lower or upper limits of core storage have been passed. This check is called an end-around check and is made at all times except for three special operations. The modification of the low-order position of core storage by − 1, except during a clear storage operation, or the modification of the high-order position of core storage by + 1, except during storage scan and storage print out operations, causes an invalid operation and a system stop.
4. Of the more than 4,000 bit configurations possible when read from a card, only 64 are *recognizable* characters. All other bit configurations are considered invalid during the data transfer from the read side of the card read-punch into core storage. A detected check condition turns on the card read validity check light. Depending on the I/O check stop switch setting on the 1447, the system also stops or a program-testable indicator is set ON.

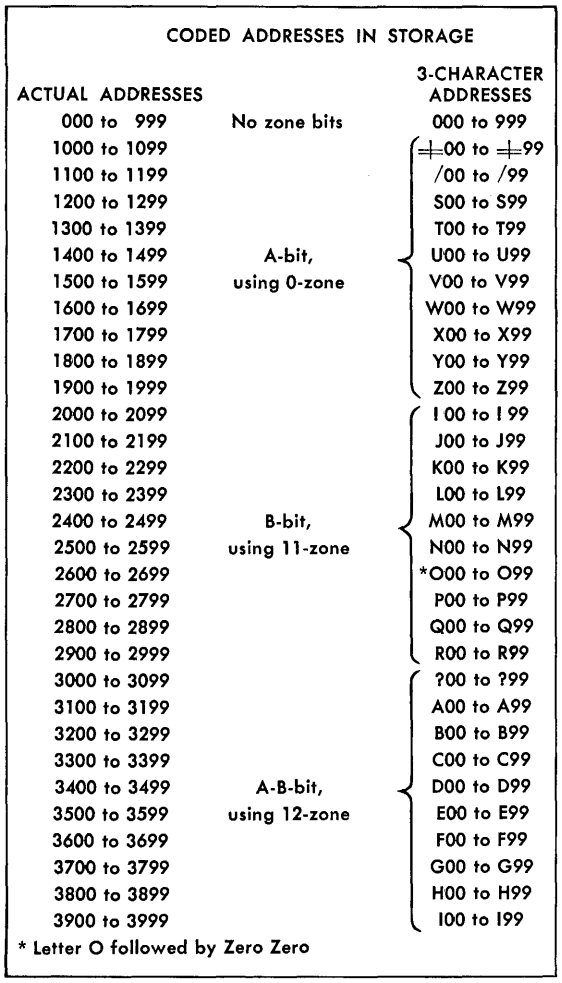
Addressing

Instructions and data used for processing in a 1440 system are kept in the core-storage area. Each core-storage position in the area has its own unique address. The IBM 1441 Processing Unit is available with four different core-storage capacities. The 1441, Model A3, contains 4,000 core-storage positions, and Model A4 contains 8,000 core-storage positions. Model A5 contains 12,000 core-storage positions, and Model A6 contains 16,000 core-storage positions.

Addressing System

Every core-storage position in the IBM 1440 Data Processing System can be addressed with a 3-character address. To address 16,000 core-storage positions with numbers only, various zone-bit configurations are added over the hundreds position and units position of the address.

The zone-bit configuration over the hundreds position specifies the thousands position of core storage up to 3999. No A- or B-bit over the hundreds position specifies that the address is the actual address (000-999). An A-bit over the hundreds position of the address specifies another group of 1,000 core-storage positions (1000-1999). A B-bit over the hundreds position of the address specifies another group of 1,000



CODED ADDRESSES IN STORAGE

ACTUAL ADDRESSES		3-CHARACTER ADDRESSES
000 to 999	No zone bits	000 to 999
1000 to 1099		≠00 to ≠99
1100 to 1199		/00 to /99
1200 to 1299		S00 to S99
1300 to 1399		T00 to T99
1400 to 1499	A-bit,	U00 to U99
1500 to 1599	using 0-zone	V00 to V99
1600 to 1699		W00 to W99
1700 to 1799		X00 to X99
1800 to 1899		Y00 to Y99
1900 to 1999		Z00 to Z99
2000 to 2099		I 00 to I 99
2100 to 2199		J00 to J99
2200 to 2299		K00 to K99
2300 to 2399		L00 to L99
2400 to 2499	B-bit,	M00 to M99
2500 to 2599	using 11-zone	N00 to N99
2600 to 2699		*O00 to O99
2700 to 2799		P00 to P99
2800 to 2899		Q00 to Q99
2900 to 2999		R00 to R99
3000 to 3099		?00 to ?99
3100 to 3199		A00 to A99
3200 to 3299		B00 to B99
3300 to 3399		C00 to C99
3400 to 3499	A-B-bit,	D00 to D99
3500 to 3599	using 12-zone	E00 to E99
3600 to 3699		F00 to F99
3700 to 3799		G00 to G99
3800 to 3899		H00 to H99
3900 to 3999		I00 to I99

* Letter O followed by Zero Zero

Figure 8. Core-Storage Address Coding

core-storage positions (2000-2999). Both the A- and the B-bit over the hundreds position of the address specify another group of 1,000 core-storage positions (3000-3999). By using these zone-bit combinations, 4,000 positions of core storage can be addressed with a 3-character address (Figure 8).

The same principle used to specify the various 1,000-blocks of core storage is also used to specify core-storage blocks of 4,000 positions. The zone-bit configuration over the units position specifies which block of 4,000 core-storage positions is being addressed.

No A- or B-bit over the units position specifies the 4,000-block in core storage that contains positions 0000-3999. An A-bit over the units position specifies the 4,000-block in core storage that contains positions 4000-7999. A B-bit over the units position specifies the 4,000-block in core storage that contains positions 8000-11999. Both the A- and the B-bit over the units position specifies the 4,000-block in core storage that contains positions 12000-15999. By combining the 3-digit address with zone-bit combinations over the hundreds and/or units position, it is possible to address 16,000 core-storage positions (Figure 9).

Data-Field Addressing

A data field in core storage is addressed by specifying the low-order (units) position of the field in the A- or B-address of the instruction. The data field is read from right to left until a word mark in the high-order position is sensed.

ACTUAL ADDRESSES	ZONE BITS OVER HUNDREDS POSITION	ZONE BITS OVER UNITS POSITION	3-CHARACTER ADDRESSES
0000 to 0999	No Zone Bits	No Zone Bits	000 to 999
1000 to 1999	A-Bit (Zero-Zone)	No Zone Bits	≠00 to Z99
2000 to 2999	B-Bit (11-Zone)	No Zone Bits	I00 to R99
3000 to 3999	AB-Bits (12-Zone)	No Zone Bits	?00 to I99
4000 to 4999	No Zone Bits	A-Bit (Zero-Zone)	00≠ to 99Z
5000 to 5999	A-Bit (Zero-Zone)	A-Bit (Zero-Zone)	≠0≠ to Z9Z
6000 to 6999	B-Bit (11-Zone)	A-Bit (Zero-Zone)	I0≠ to R9Z
7000 to 7999	AB-Bits (12-Zone)	A-Bit (Zero-Zone)	?0≠ to I9Z
8000 to 8999	No Zone Bits	B-Bit (11-Zone)	00I to 99R
9000 to 9999	A-Bit (Zero-Zone)	B-Bit (11-Zone)	≠0I to Z9R
10000 to 10999	B-Bit (11-Zone)	B-Bit (11-Zone)	I0I to R9R
11000 to 11999	AB-Bits (12-Zone)	B-Bit (11-Zone)	?0I to I9R
12000 to 12999	No Zone Bits	AB-Bits (12-Zone)	00? to 99I
13000 to 13999	A-Bit (Zero-Zone)	AB-Bits (12-Zone)	≠0? to Z9I
14000 to 14999	B-Bit (11-Zone)	AB-Bits (12-Zone)	I0? to R9I
15000 to 15999	AB-Bits (12-Zone)	AB-Bits (12-Zone)	?0? to I9I

Figure 9. 1440 Addressing System

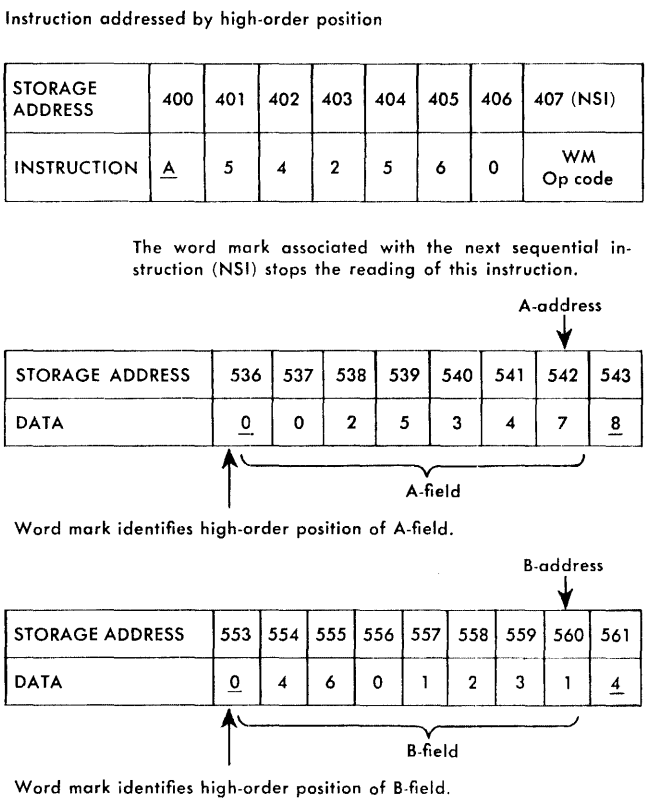


Figure 10. Data and Instruction Addressing

Instruction Addressing

An instruction in core storage is addressed by giving the high-order (operation code) position of the instruction. All operation codes must have a word mark. (This word mark is normally set by the loading routine when the instructions are loaded.) The machine reads an instruction from left to right until it senses the word mark associated with the next sequential instruction. The final instruction in the program must have a word mark set at the right of its low-order position. (The word mark is not needed if the instruction is UNCONDITIONAL BRANCH, SET WORD MARK, OR CLEAR STORAGE.)

Example: Instruction address 400 (Figure 10) contains the operation code for the following instruction:

Op Code	*A-address*	*B-address*
A	542	560

When this instruction is executed, the data in the A-field is added to the data in the B-field:

```
  0025347
  04601231
 ---------
  04626578
```

The result is stored in the B-field.

Core-Storage Area Assignment

There are two areas in core storage that are used for specific purposes. Core-storage positions 001-081 are used in conjunction with a program-load operation and core-storage positions 087-089, 092-094, and 097-099 are used as three index registers when the indexing and store address register special feature is used. All other core-storage positions are always available for normal use, and the areas just mentioned can be used for other system operations when they are not being used as specified.

1440 Register Operation

The IBM 1440 Data Processing System operates on and processes data to produce a desired result by executing a series of instructions. A series of instructions designed to solve a problem is known as a *program.* Because these instructions are retained in core storage, it is more properly called a stored program.

The processing unit must interpret an instruction and perform the function prescribed by the instruction. To do this, various types of devices that are capable of receiving information, storing it, and transferring it as directed by control circuits are used. These devices are known as *registers.* The 1440 has seven registers, four are address registers and three are character registers (Figure 11).

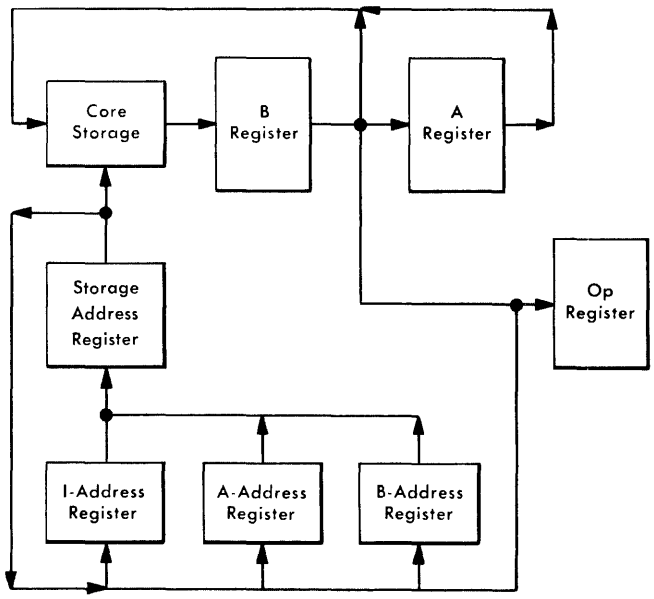


Figure 11. Processing Unit Registers

ADDRESS REGISTERS

There are four address registers in the IBM 1441 Processing Unit. One register controls the program sequence, and two other registers control the data transfer from one storage location to another. The fourth register specifies which storage location is active during a particular storage cycle.

I-Address Register. The I- (Instruction) address register always contains the storage location of the next instruction character to be used by the stored program. The number in this register is increased by one as the instruction is read from left to right.

A-Address Register. The A-address register contains the storage address of the data in the A-address portion of an instruction. Normally, as the instruction is executed, the number in this register is decreased by 1 after each storage cycle that involves the A-address.

NOTE: If the A-address portion of the instruction does not contain a core-storage address (for example %Gx) the contents of the A-address register are not altered as the instruction is executed.

B-Address Register. This register contains the storage location of the data in the B-address portion of an instruction. Normally, as a storage cycle involving the B-address is executed, the storage address in the B-address register is decreased by 1.

Storage-Address Register. The storage-address register always contains the address of the core-storage position that will be involved in any data movement during that particular machine cycle.

CHARACTER REGISTERS

The A- and B-character registers and the Op-register are single-character registers used to store data during the execution of an instruction.

Op-Register. The Op- (Operation) register stores the operation code of the instruction in process for the duration of the operation. The operation code is stored in BCD code, including the check bit but excluding the word mark.

B-Register. Each character leaving core storage enters the B-register. The character is stored in 8-bit form (BCD code, check bit, and word mark). The B-register is reset and filled with a character from core storage on every storage cycle.

A-Register. The A-register is reset and filled with the character from the B-register during each storage cycle that involves the A-address, and during all instruction cycles except the first and last I- (Instruction) cycle of each instruction. Data is stored in 8-bit form

NOTE: Information can be written back into core storage directly from either the A- or B-register.

Figure 12 shows the I-phase of an operation and gives a detailed schematic for loading a 7-character instruction in the operation-code register, in the A- and B-registers and in the I-, A-, and B-address registers. Eight storage cycles are required to load the complete instruction in the register. Each storage cycle requires .0111 ms.

NOTE: The A- and B-address registers contain 3-character addresses. The addresses shown in this schematic are 4-digit addresses because the storage display lights on the console show 4-digit addresses. Refer to Figure 8 for the relationship between 3- and 4-digit addresses.

Chaining Instructions

In some programs, it is possible to perform a series of operations on several fields that are in consecutive storage locations. Some of the basic operations, such as add, subtract, move, and load, can be *chained* so that less time is required to perform the operations, and space is saved in storing instructions. Here is an example of the chaining technique: assume that four 5-position fields stored in sequence are to be added to four other sequential fields. This operation could be done using four 7-character instructions:

```
A  700  850
A  695  845
A  690  840
A  685  835
```

At the completion of the first instruction, the A-address register contains 695 and the B-address register contains 845. These are the same numbers that are in the A- and B-addresses in the second instruction. (Executing the second and third instructions also results in A- and B-addresses that are the same as the A-and B-addresses of the third and fourth instructions.) Eighty storage cycles would be required to execute these instructions, thus using up .888 ms. Also, 28 storage positions are required to store these instructions.

By taking advantage of the fact that the A- and B-address registers contain the necessary information to perform the next instruction, this same sequence of operations can be executed as follows:

```
A  700  850
A
A
A
```

CYCLE	OPERATION	Instruction Location: A 197 | 5 198 | 6 199 | 7 200 | T 201 | 1 202 | 2 203 | S 204
I-Op	The operation code enters the B-register and the Op-register. Because this is the first I-cycle, the A-register is undisturbed.	I Register: 0|1|9|7 — B Register: A — A Register: ? — Cycle 1 OP Register: A — A Address Register: ?|?|?|? — B Address Register: ?|?|?|?
I-1	The A-address register is reset to blanks during the first part of the cycle for all instructions. The B-address register is reset to blanks during the first part of the cycle for all operations except Move, Load, Store A- and Store B-address Register operation. During the I-1 cycle, the second instruction character (first character of the A-address) enters the thousands and hundreds positions of the A- and B-address registers and the A-register by the way of the B-register.	I Register: 0|1|9|8 — B Register: 5 — A Register: 5 — Cycle 2 OP Register: A — A Address Register: 0|5|b|b — B Address Register: 0|5|b|b
I-2	The third character of the instruction enters the tens position of the A- and B-address registers, and the A-register through the B-register.	I Register: 0|1|9|9 — B Register: 6 — A Register: 6 — Cycle 3 OP Register: A — A Address Register: 0|5|6|b — B Address Register: 0|5|6|b
I-3	The fourth instruction character enters the units position of the A- and B-address registers, and the A-register through the B-register.	I Register: 0|2|0|0 — B Register: 7 — A Register: 7 — Cycle 4 OP Register: A — A Address Register: 0|5|6|7 — B Address Register: 0|5|6|7
I-4	The B-address register is reset at the beginning of this cycle. The fifth instruction character (first character of the B-address) enters the hundreds position of the B-address register, and the A-register through the B-register.	I Register: 0|2|0|1 — B Register: T — A Register: T — Cycle 5 OP Register: A — A Address Register: 0|5|6|7 — B Address Register: 1|3|b|b
I-5	The sixth instruction character goes to the tens position of the B-address register, and the A-register through the B-register.	I Register: 0|2|0|2 — B Register: 1 — A Register: 1 — Cycle 6 OP Register: A — A Address Register: 0|5|6|7 — B Address Register: 1|3|1|b
I-6	The seventh character of the instruction (last character of the B-address) enters the units position of the B-address register and the A-register through the B-register.	I Register: 0|2|0|3 — B Register: 2 — A Register: 2 — Cycle 7 OP Register: A — A Address Register: 0|5|6|7 — B Address Register: 1|3|1|2
I-7	The first character of the next instruction enters the B-register only. Because this is the last I-cycle for this instruction, the A-register and the Op-register, the A- and B-address registers are undisturbed. The detection of a word mark associated with this character signals the machine that this is the Op code for the next instruction. The loading operations stops, and the instruction that was just loaded is executed. Note that the I-address register contains the address of the high-order position of the next sequential instruction.	I Register: 0|2|0|4 — B Register: S — A Register: 2 — Cycle 8 OP Register: A — A Address Register: 0|5|6|7 — B Address Register: 1|3|1|2

Figure 12. Instruction Loading Schematic

Connecting instructions together in this manner is called *chaining.* The first add instruction contains both the A- and B-addresses. The following three instructions contain only the operation code for those instructions. The A- and B-addresses are the results left in the A- and B-address registers from the previous instruction. This type of operation requires 62 storage cycles, and takes .688 ms to execute. Storing these chained instructions requires only ten storage positions.

The ability to chain a series of instructions does not depend on the use of the same operation code. Chained instructions may have various Op codes. To be operated on, the A-fields must be in sequence, and the B-fields must be in sequence. *Example:*

A 900 850
M
A
M

Assume that the data fields are each ten characters long:

The ten characters at location 900 were added to 850.
The ten characters at location 890 were moved to 840.
The ten characters at location 880 were added to 830.
The ten characters at location 870 were moved to 820.

The description of each instruction includes the contents of the address registers after the operation has been performed. Figure 4 shows the abbreviations that indicate the contents of these registers.

By using this information, the programmer can determine the status of the registers and decide whether chaining is practical in specific cases.

NOTE: Instructions that don't contain core-storage addresses cannot be chained. For example, M %Gn xxx R is a READ CARD instruction. The card read-punch is signaled as the machine reads the instruction. Although the A-address register contains %7n after the operation, chaining is impossible because the machine does not select the unit from the contents of the A-address register.

Most single-address instructions Op code and an A-address) cause the A-address to be inserted in both the A-address and B-address registers (for example, A xxx. However, executing a MOVE, LOAD, or a STORE B-ADDRESS REGISTER instruction does not disturb the B-address register, and permits the programmer to use the previous contents of that register as part of the instruction.

All no-address instructions (Op code and I-address) depend on whether the indexing and store address register special feature is installed on the system:

1. With the special feature installed, the B-address register contains the address of the next sequential instruction, if a branch occurs.
2. Without the special feature installed, the B-address register is cleared to blanks whenever a branch occurs.

Address Modification

It becomes necessary in some 1440 programs to perform the same operations repetitively, with a change only in the A- or B-address. Changing of an address while retaining the rest of the instruction is called *address modification.* Address modification can result in savings in the number of program steps and in the number of storage requirements. In some cases, the program itself determines if, and how, addresses are to be changed to perform the correct program steps for conditions arising during data processing.

The methods that can be used to modify addresses on a specific system depend on the core-storage capacity of that system.

On 1440 systems equipped with 4,000 positions of core storage, address modification is accomplished by either using modulus 4 arithmetic or installing the indexing and store address register special feature.

On 1440 systems equipped with more than 4,000 positions of core storage, the two previously mentioned methods of address modification can be used. Also, these systems have a MODIFY ADDRESS instruction that greatly simplifies address modification.

Modulus 4 Arithmetic Method

When modifying addresses by modulus 4 arithmetic, the modified address should be located in the same 4,000-block of core storage as the original address. This is because a zone-bit overflow of over three in the hundreds position of the address cannot be transferred to the units position of the address.

To set up a workable modulus 4 system, these digital values are assigned the four possible zone-bit configurations that appear in the hundreds position:

No A-, No B-bit = 0
A-bit = 1
B-bit = 2
A- and B-bit = 3

As can be seen, the highest possible digit is three. Values in excess of three are equal to that value minus

A + A	=	B	or	1 + 1 = 2
A + B	=	AB	or	1 + 2 = 3
B + B	=	NoANoB	or	2 + 2 = 0
A + AB	=	NoANoB	or	1 + 3 = 0
A + NoANoB	=	A	or	1 + 0 = 1
B + AB	=	A	or	2 + 3 = 1
B + NoANoB	=	B	or	2 + 0 = 2
AB + AB	=	B	or	3 + 3 = 2

Figure 13. A-Bit and B-Bit Values

four. For example, a value of five is represented as a value of 1 (Figure 13).

Address modification to a higher address in the 000-999 address range is:

Increase address 472 by 345

472 + 345 = 817

This is a normal add operation with no overflow involved.

Address modification to an address greater than 1000 is:

Increase address 912 by 314

912 + 314 = 1226 or S 26

S = A2 *(overflow in high-order position sets an A-bit using modulus 4 arithmetic and turns on the arithmetic overflow indicator).*

Increase address 1754 (X54) by 1204 (S04)

1754 + 1204 = 2958

X54 + S04 = R58

X = (A7)

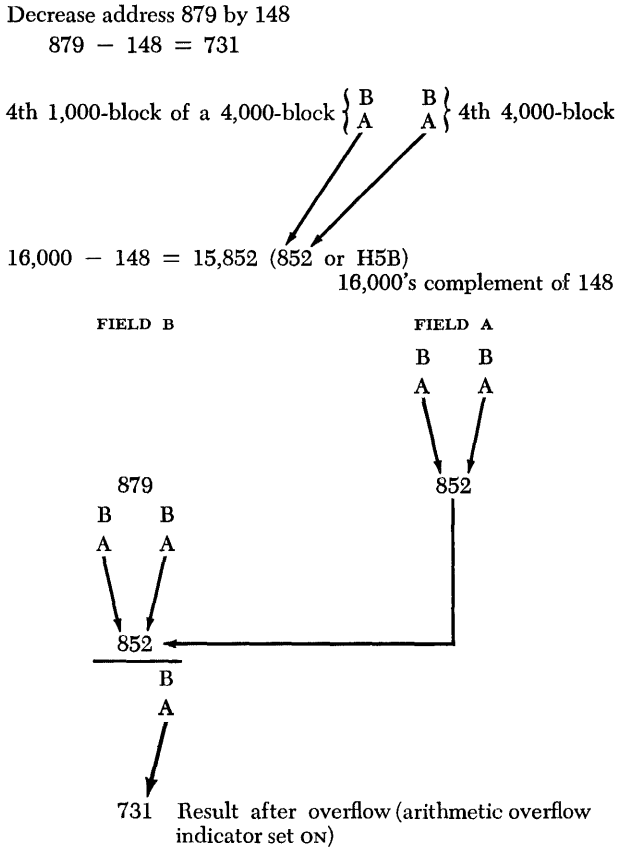
S = (A2)

Using the rules of modulus 4 arithmetic, A + A = B-bit, the new address is:

958 with a B-bit over the high-order position (B9 = R) or R58 (2958).

To decrease an address, a different means must be used. Modulus 4 arithmetic operates for addition only. Decreasing an address requires the addition of a complement, rather than doing a conventional subtract operation.

In systems equipped with 4,000 core-storage positions, the 16,000's complement of the decrement figure is added to the address to be modified (modulus 16 arithmetic).



The add operation is performed as shown. The A-field figure is added to the B-field figure. The digital result is 731 and the arithmetic overflow indicator is set ON. Because an add operation has taken place, the units position ends up with a plus sign (an A- and a B-bit). The arithmetic overflow in the hundreds position adds an A-bit to the A- and B-bits already there, resulting in a zone-bit configuration of no A- and no B-bit (see Figure 13). The A-bit addition increases the zone-bit value to 16. A value of 16, according to modulus 16 rules, has a new address value of 0 (000-999 core-storage address block). This means that 731 is the actual address.

Modulus 4 arithmetic is normally used in 1440 systems that contain only 4,000 core-storage positions. With care, this address modification method could be used on systems with more core-storage capacity, but its usefulness is negligible because 1440 systems with more than 4,000 core-storage positions are equipped with the MODIFY ADDRESS instruction.

Modify Address Instruction Method

IBM 1440 systems with more than 4,000 core-storage positions can easily modify any address by using the MODIFY ADDRESS instruction.

Modify Address (Two Addresses)

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*
MA	#	xxx	xxx

Function. This instruction causes the 3-character field, specified by the A-address (A-field), to be added to the 3-character field specified by the B-address (B-field). The result is stored in the B-field. The three numerical portions and the zones of the units and hundreds positions of the B-field make up the 3-character result. For example:

Location	*Contents*	*3-Character Address*	*Actual Address*
A-address	A-field	100	100
B-address	B-field	L2F	14326
	B-field	M2F	14426

Word Marks. Word marks are not affected, and are not required to define the A- or B-fields. If word marks are present, they are ignored and remain unchanged in both fields.

Timing. $T = .0111\ (L_I + 9)$ ms.

Note: Rules for the addition of zone bits are the same as in modulus 4 arithmetic, with one addition. This instruction makes it possible to reflect the hundreds position zone-bit overflow in the units position when the address is modified to a higher 4,000-block of core storage. When a zone-bit overflow occurs during the hundreds position modification, an additional cycle is executed to adjust the units position zone-bit configuration.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-3	B-1 or B-3

Example. Add the 3-character address labeled ADDA (0985) to the 3-character address labeled ADDB (1313), Figure 14.

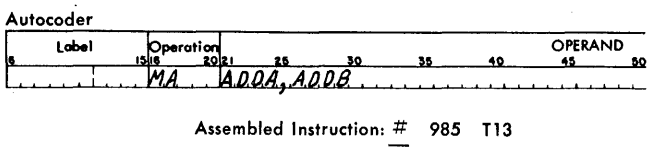


Figure 14. Modify Address (Two-Addresses)

Modify Address (One Address)

Instruction Format.

Mnemonic	*Op Code*	*A-address*
MA	#	xxx

Function. This format of the MODIFY ADDRESS instruction causes the 3-character field, specified by the A-address, to be added to itself. The result is stored in the A-field.

Word Marks. Word marks are not required to define the A-field. If they are present, they are ignored and remain undisturbed in the A-field.

Timing. $T = .0111\ (L_I + 9)$ ms.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-3	A-1 or A-3

Example. Double the address labeled ADDC (2956), and store the result at ADDC (Figure 15).

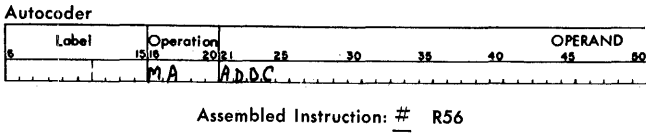


Figure 15. Modify Address (One-Address)

Indexing Method

Any 1440 system can modify addresses by installing the indexing and store address register special feature. A complete description of this feature can be found in *Special Features,* Form A26-5669.

The operations performed by an IBM 1440 Data Processing System can be arranged into these general classifications:

1. Arithmetic operations
2. Logic operations
3. Data-moving operations
4. Miscellaneous operations
5. Edit operation
6. IBM 1447 Console operations

Arithmetic Operations

The IBM 1440 Data Processing System adds and subtracts, by applying the add-to-storage method of operation. The two factors to be combined are added within core storage without the use of special accumulators or counters. Because any storage area can be used as an accumulator field, the capacity for performing arithmetic functions is not limited by standard-size accumulators or by a predetermined number of accumulators within the system. In arithmetic operations, the 1440 system considers blanks and zeros the same. An unsigned field is considered positive by the system.

All arithmetic functions are performed under complete algebraic sign control. The sign of a factor is determined by the combination of zone bits in the units position of the fields specified by the instruction being executed.

Figure 16 shows the four possible combinations of zone bits and the values of the signs they represent.

The standard machine method of signing a field is to indicate a positive factor with A- and B-bits (12-zone), and to indicate a negative factor with a B-bit (11-zone).

The arithmetic operations in the IBM 1440 Data Processing System are performed by using one of two types of add cycles incorporated in the system. The two types of add cycles are:

1. true add
2. complement add

The type of add cycle performed depends on the arithmetic operation and the signs and values of the two factors involved (Figure 17).

SIGN	BCD CODE BIT CONFIGURATION	CARD CODE CONFIGURATION
Plus	No A- or B-Bit	No Zone
Plus	A- and B-Bits	12 Zone
Minus	B-Bit Only	11 Zone
Plus	A-Bit Only	0 Zone

Figure 16. Sign Bit Equivalents

TYPE OF OPER.	A-FLD. SIGN	B-FLD. SIGN	TYPE OF ADD CYCLE	SIGN OF RESULT
ADD	+	+	True Add	+
		−	Compl. Add	Sign of Greater
	−	+	Compl. Add	Value
		−	True Add	−
SUBTRACT	+	−	True Add	−
		+	Compl. Add	Sign of Greater
	−	−	Compl. Add	Value
		+	True Add	+

Figure 17. Types of Add Cycles and Sign of Result for Add and Subtract Operations

True Add

A true-add cycle is specified when the total number of minus signs is an even number (0 or 2). The signs considered are the signs of the factors and the sign of the operation.

The sign of the result after a true-add cycle carries the original sign of the B-field when either an add or a subtract operation is performed (Figure 18).

Complement Add

An uneven number of minus signs (1 or 3) specifies a complement-add cycle. The system converts the A-field factor to its nines complement figure and adds it to the B-field factor (plus one initial carry). The system then initiates a carry test to determine whether a carry occurred from the high-order position of the

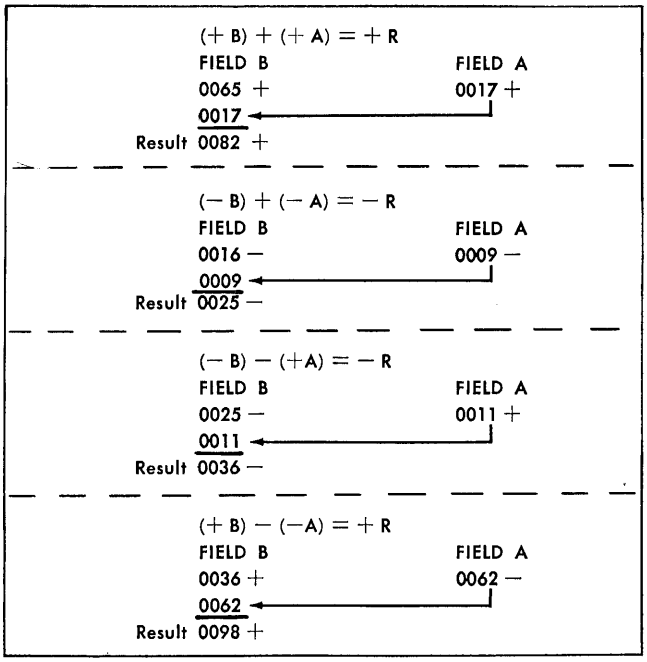


Figure 18. True-Add Cycle Examples

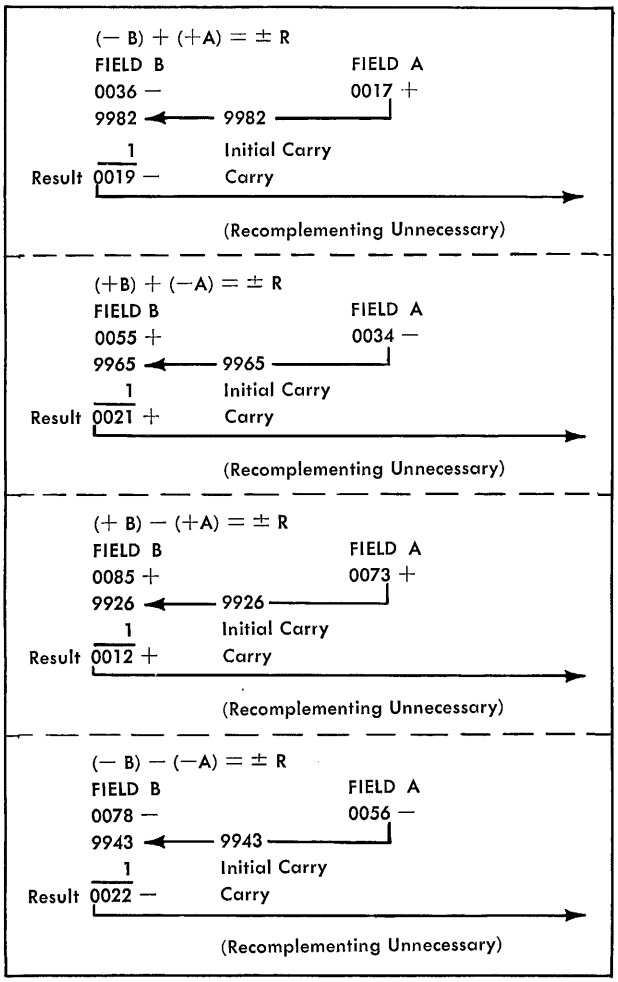


Figure 19. Complement-Add Cycle Examples

B-field. The presence of a carry indicates that the result in the B-field is a true figure (Figure 19). The original sign of the B-field is the sign of the result.

If there was no carry from the high-order position of the B-field, the result in the B-field is not a true figure. A recomplement cycle is performed to convert the result to a true figure. In an add operation that results in a negative figure, the sign of the result is always changed during a recomplement cycle, (Figure 20). The system generates the new sign automatically. A positive factor is indicated by the presence of an A- and B-bit over the units position of the factor. After a complement-add cycle, the sign of the result carries the sign of the greater value factor.

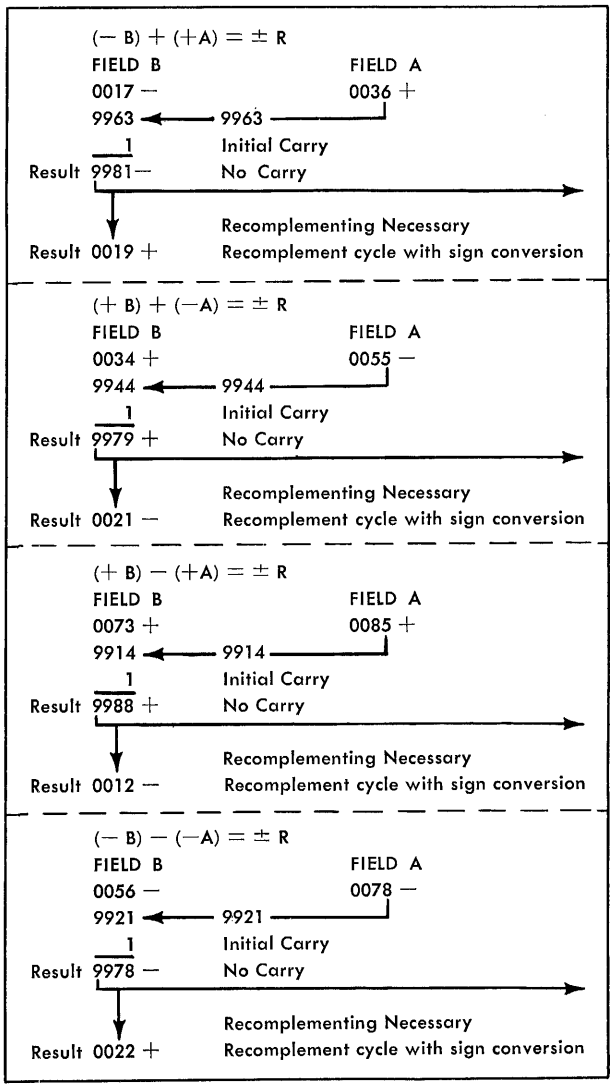


Figure 20. Complement-Add (with Recomplementing) Cycle Examples

Arithmetic Instructions

Add (Two Fields)

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*
A	A	xxx	xxx

Function. The data in the A-field is added algebraically to the data in the B-field. The result is stored in the B-field.

Word Marks. The B-field must have a defining word mark, because it is this word mark that actually stops the add operation.

The A-field must have a word mark, only if it is shorter than the B-field. In this case, the transmission of data from the A-field stops after the A-field word mark is sensed. Zeros are then inserted in the A-register until the B-field word mark is sensed.

If the A-field is longer than the B-field, the high-order positions of the A-field that exceed the limits imposed by the B-field word mark are not processed. For overflow conditions and considerations, assume that the A-field is the same length as the B-field. (See *Address Modification.)*

Timing.

1. If the operation does not require a recomplement cycle:
   $T = .0111 (L_I + I + L_A + L_B)$ ms.
2. If a recomplement cycle is taken:
   $T = .0111 (L_I + I + L_A + 3\ L_B)$ ms.

If the multiply-divide special feature is installed, the 1440 timing for a recomplement cycle is:

$T = .0111 (L_I + 1 + L_A + 2\ L_B)$ ms.

Notes.

1 *Sign control* (see Figure 17):

If a recomplement cycle is taken, the sign of the B- (result) field is changed and the result is stored in true form.

2. *Zone bits:*

If the fields to be added contain zone bits in other than the high-order position of the B-field and the sign positions of both fields, only the digits are used in a true-add operation. B-field zone bits are removed except for the units and high-order positions in a true-add operation. If a complement add takes place, zone bits are removed from all but the units positions of the B-field.

3. *Overflow indication:*

If an overflow occurs during a true-add operation, the overflow indicator is set ON, and the overflow indications are stored over the high-order digit of the B-field. When the A-field exceeds, or is equal to, the B-field length, and the A-field position that corresponds to the high-order B-field position contains a zone bit, this zone bit is added to any zone bits present in the high-order B-field position.

Condition	*Result*
First overflow	A-bit
Second overflow	B-bit
Third overflow	A- and B-bits
Fourth overflow	No A- or B-bits

For subsequent overflows repeat conditions 1 through 4. Overflow indication does not occur for a 1-position field.

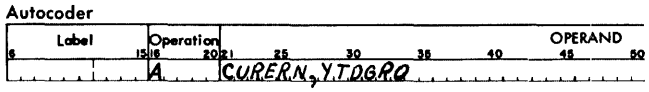
The BRANCH IF ARITHMETIC OVERFLOW INDICATOR ON, B (III) Z, instruction tests and turns off the overflow indicator, and branches to an instruction or group of instructions if an overflow condition occurred. There is only one overflow indicator in the system. It is turned off either by executing a BRANCH IF ARITHMETIC OVERFLOW INDICATOR ON instruction or pressing the start reset key on the 1447 operator panel.

Overflow indication does not occur for a 1-position field.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-L_W	B-L_B

Example. Add CURERN (0506) to YTDGRO (0708), Figure 21.



Assembled Instruction: A 506 708

Figure 21. Add (Two Fields)

Add (One Field)

Instruction Format.

Mnemonic	*Op Code*	*A-address*
A	A	xxx

Function. This format of the ADD instruction causes the data in the A-field to be added to itself.

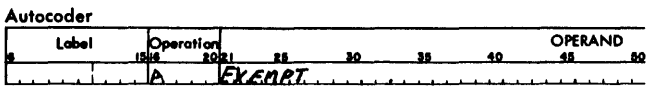
Word Marks. The A-field must have a defining word mark. It is this word mark that stops the add operation. This instruction must be followed by a word mark in the position after the A-address.

Timing. $T = .0111 (L_I + 1 + 2\ L_A)$ ms.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-L_A	A-L_A

Example. Add to itself the data at EXEMPT (0981), Figure 22.



Assembled Instruction: A 981

Figure 22. Add (One Field)

Subtract (Two Fields)

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*
S	S	xxx	xxx

Function. The numerical data in the A-field is subtracted algebraically from the numerical data in the B-field. The result is stored in the B-field. Refer to Figure 17 for the sign that results from a specific subtract operation.

Word Marks. A word mark is required to define the B-field. An A-field requires a word mark, only if it is shorter than the B-field. In this case, the A-field word mark stops transmission of data from the A-field.

Timing.

1. If the operation does not require a recomplement cycle:
   $T = .0111\ (L_I + 1 + L_A + L_B)$ ms.
2. Subtract – recomplement cycle necessary:
   $T = .0111\ (L_I + 1 + L_A + 3L_B)$ ms.

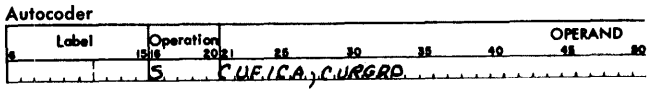
   If the multiply-divide special feature is installed, the 1440 timing for a recomplement cycle is:
   $T = .0111\ (L_I + 1 + L_A + 2L_B)$ ms.

Note. If a recomplement cycle is taken, the sign of the B-(result) field is changed, and the result is stored in true form.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-L_W	B-L_W

Example. Subtract CUFICA (00753) from CURGRO (0896), Figure 23.



Assembled Instruction: S 753 896

Figure 23. Subtract (Two Fields)

Subtract (One Field)

Instruction Format.

Mnemonic	*Op Code*	*A-address*
S	S	xxx

Function. The data at the A-address is subtracted from itself. If the A-field sign is minus, the result is a minus zero. If the A-field sign is plus, the result is a plus zero.

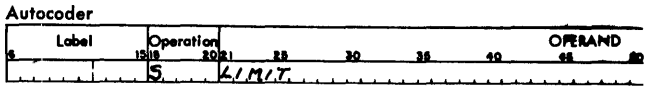
Word Marks. The A-field must have a defining word mark. This instruction must be followed by a word mark in the position after the A-address.

Timing. $T = .0111\ (L_I + 1 + 2L_A)$ ms.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-L_A	A-L_A

Example. Subtract from itself the field labeled LIMIT (units position is 0395), Figure 24.



Assembled Instruction: S 395

Figure 24. Subtract (One Field)

Zero and Add (Two Fields)

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*
ZA	?	xxx	xxx

Function. This instruction functionally adds the A-field to a *zeroed* B-field. Technically, this is accomplished by moving the A-field to the B-field. The high-order

positions of the B-field are set to zero if the B-field is larger than the A-field. The data from the A-field moves directly from the A-register to storage. Zone bits are stripped from all positions except the units position. Blanks in the A-field are stored as blanks the B-field.

Word Marks. A word mark is required for definition of the B-field. It is required in the A-field, only if it is shorter than the B-field. If the A-field is shorter than the B-field, all extra high-order B-field positions contain zeros. But the transmission of data from A stops when the A-field word mark is detected.

Timing. $T = .0111 (L_I + 1 + L_A + L_B)$ ms.

Note. The sign of the result always has both A- and B-bits if it is positive. If the sign is negative, it has only a B-bit.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-L_W	B-L_B

Example. Zero WHTAX area (0796-0802) and add new TAX (0749-0754) to WHTAX (Figure 25).

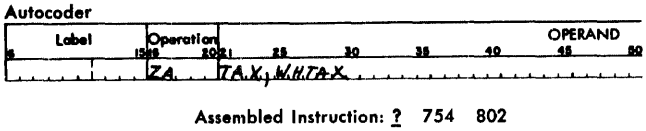


Figure 25. Zero and Add (Two Fields)

Zero and Add (One Field)

Instruction Format.

Mnemonic	*Op Code*	*A-address*
ZA	?	xxx

Function. This format of the ZERO AND ADD instruction is used to strip the A-field of all zone bits, except in the units (sign) position. The A-field sign is retained. If the A-field plus sign bit configuration is not an A- and B-bit, it is changed to the A- and B-bit configuration.

Word Marks. The A-field must have a word mark in its high-order position.

Timing. $T = .0111 (L_I + 1 + 2L_A)$ ms.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-L_A	A-L_A

Example. Strip zone bits from TOTAL (0560) area (Figure 26).

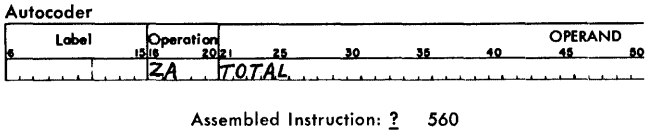


Figure 26. Zero and Add (One Field)

Zero and Subtract (Two Fields)

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*
ZS	!	xxx	xxx

Function. This instruction functionally subtracts the A-field from a *zeroed* B-field. Technically, this is accomplished by moving the A-field to the B-field. The high-order positions of the B-field are set to zero if the B-field is moved directly from the A-register to the B-field. Zone bits are stripped from all but the sign (units) position. The sign is represented in standard form.

Word Marks. A word mark is required to define the B-field. If the A-field is shorter than the B-field, the A-field must have a defining word mark to stop transmission of data to B. The extra high-order B-field positions contain zeros, if A is shorter than B.

Timing. $T = .0111 (L_I + 1 + L_A + L_B)$ ms.

Note. If the A-field is positive, the B-field result is negative. If the A-field is negative, the B-field result is positive.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-L_W	B-L_B

Example. Zero ACCUM 1 (0755) and subtract TAXEXP (0699) from ACCUM 1, Figure 27.

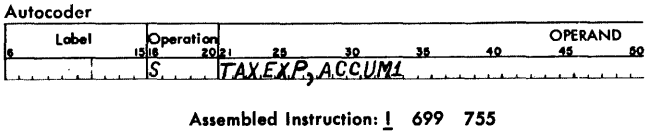


Figure 27. Zero and Subtract (Two Fields)

Zero and Subtract (One Field)

Instruction Format.

Mnemonic	*Op Code*	*A-address*
ZS	!	xxx

Function. This instruction changes the A-field sign, and strips all A-field zone bits, except in the units (sign) position.

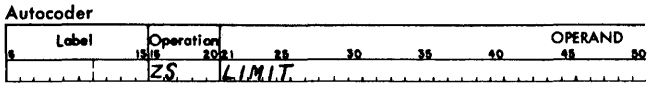
Word Marks. The data in the A-field requires a word mark in its high-order position.

Timing. $T = .0111\ (L_I + 1 + 2L_A)$ ms.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	$A\text{-}L_A$	$A\text{-}L_A$

Example. Subtract LIMIT (0495) from zero, and change sign of LIMIT's value (Figure 28).



Assembled Instruction: ! 495

Figure 28. Zero and Subtract (One Field)

Logic Operations

The 1440 program can test for certain conditions that may arise during processing, and can transfer the program to a predetermined set of instructions or subroutines, as a result of these specific tests. The operations that perform these testing operations are called logic operations.

For example, if an overflow occurs in an arithmetic operation, a routine to handle this condition can be initiated by executing a BRANCH IF ARITHMETIC OVERFLOW INDICATOR ON instruction. Branching to this routine is called a *conditional branch.* The sequential execution of program steps is bypassed, and the program branches to the address of the instruction specified by the I-address of this conditional branch instruction. If the condition had not been present, the system would have started reading the instruction that appears at the immediate right of the conditional branch instruction (next sequential instruction). All conditional branch instructions have a d-character that is used to specify the conditions necessary for a program transfer.

A branch that occurs as a direct result of the execution of the instruction itself is called an *unconditional branch.* No special condition (other than the execution of the program step) is needed to transfer the program out of its normal sequential execution.

Any branch operation that terminates with a successful branch to another portion of core storage for the next instruction address operates as follows:

- The B-address register is reset to blanks during the next instruction operation (I-Op) cycle.
- If the indexing and store address register special feature is installed on the system, the next sequential instruction (NSI) is placed in the B-address register and during the following instruction the B-address register is not set to blanks.

Logic Instructions

Branch (Unconditional)

Instruction Format.

Mnemonic	*Op Code*	*I-address*
B	B	III

Function. This instruction always causes the program to branch to the address specified by the I-address position of the instruction. This address contains the Op code of some instruction.

This unconditional branch operation is used to interrupt normal program sequence, and to continue the program at some other desired point, without testing for specific conditions.

Word Marks. The instruction is executed correctly if the core-storage position next to the I-address units position contains either a blank or a word mark.

Timing.

Branch (without indexing):

$T = .0111 (L_I + 1)$ ms.

Branch (with indexing):

$T = .0111 (L_I + 2)$ ms.

Address Registers After Operation.

	I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
Branch (without indexing)	NSI	BI	blank
Branch (with indexing)	NSI	BI	NSI

Example. Unconditionally branch to AGAIN (3498), Figure 29.

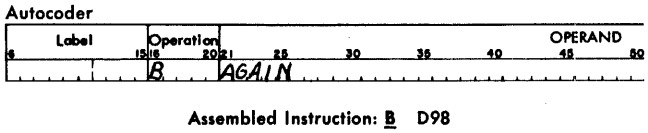


Figure 29. Branch (Unconditional)

Branch If Indicator On

Instruction Format.

Mnemonic	*Op Code*	*I-address*	*d-character*
See Figure 30.	B	xxx	x

Function. The d-character specifies the indicator tested. If the indicator is on, the next instruction is taken from the I-address. If the indicator is off, the next sequential instruction is taken. Figure 30 shows the valid d-characters, the indicators they test, and the conditions that turn the indicators off.

Word Marks. Word marks are not affected.

Timing.

No Branch:

$T = .0111 (L_I + 1)$ ms.

Branch (without indexing):

$T = .0111 (L_I + 1)$ ms.

Branch (with indexing):

$T = .0111 (L_I + 2)$ ms.

MNEMONIC	d CHARACTER	BRANCH ON	RESET BY
BC9	9	Carriage Channel #9	Branch Test or Channel 1 punch
BCV	@	Carriage Channel #12	
BPB	P	* Printer Busy	Machine Circuitry
BLC	A	"Last Card" switch (sense switch A)	Manual System Operator (Switch) or next card feed cycle
BSS+	B	* Sense Switch B	System Operator
BSS+	C	* Sense Switch C	
BSS+	D	* Sense Switch D	
BSS+	E	* Sense Switch E	
BSS+	F	* Sense Switch F	
BSS+	G	* Sense Switch G	
BAV	Z	Arithmetic Overflow	Branch Test
BIN+	%	Processing Check with Check Stop Switch Off	
BIN+	?	Read Error	Reset by Branch Test
BIN+	!	Punch Error } If I/O Check Stop switch is off	
BIN+	#	Printer Error	
BIN+	N	Access Inoperable	Next Disk Storage operation
BIN+	\ (left oblique)	Access Busy	
BIN+	V	Disk Error	
BIN+	W	Wrong-Length Record	
BIN+	X	Unequal-Address Compare	
BIN+	Y	Any-Disk Condition	
BU	/ (diagonal)	Unequal Compare (B ≠ A)	Next Compare or Disk Storage operation
BE	S	Equal Compare (B = A)	
BL	T	Low Compare (B < A)	
BH	U	High Compare (B > A)	

* Special Feature

\+ d-character must be coded in operand portion of instruction

Figure 30. Branch if Indicator On Mnemonics, d-Characters, and Conditions

Address Registers After Operation. All d-characters.

	I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
No Branch	NSI	BI	dbb
Branch (without indexing)	NSI	BI	blank
Branch (with indexing)	NSI	BI	NSI

Example. Test for last card. If it is the last card, branch to END (0599), Figure 31.

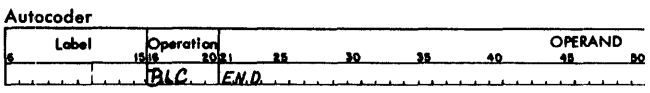


Figure 31. Branch if Indicator On

Branch If Character Equal

Instruction Format.

Mnemonic	*Op Code*	*I-address*	*B-address*	*d-character*
BCE	B	xxx	xxx	x

Function. This instruction causes the single character at the B-address to be compared to the d-character. If the comparison is equal, the program branches to the I-address for the next instruction. If the two characters are not the same, the program continues with the next sequential instruction.

Word Marks. Word marks in the location tested have no effect on the operation.

Timing.

No Branch:

$T = .0111 (L_I + 2)$ ms.

Branch (without indexing):

$T = .0111 (L_I + 2)$ ms.

Branch (with indexing):

$T = .0111 (L_I + 3)$ ms.

Address Registers After Operation.

	I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
No Branch	NSI	BI	B-1
Branch (without indexing)	NSI	BI	blank
Branch (with indexing)	NSI	BI	NSI

Example. This example shows how the chaining method can be used to test an entire field for blank characters. Each position in the area labeled AMOUNT (0350, 0349, 0348 and 0347) is individually tested for a blank character. If a blank is found, the program branches to BLANK (0601) for the next instruction. If the position tested contains a character, the program continues in sequence (Figure 32).

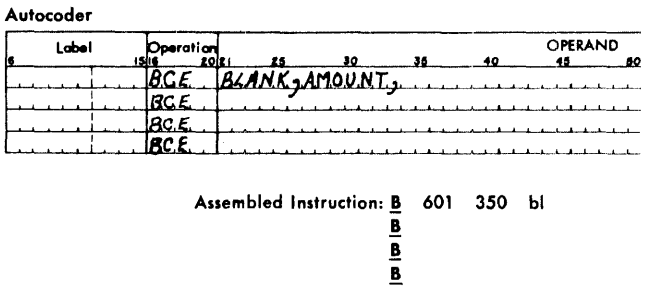


Figure 32. Branch if Character Equal

MNEMONIC	D CHARACTER	CONDITION
BW	1	Word mark
BWZ	2	No zone (No A, No B bit)
BWZ	B	12 zone (AB bits)
BWZ	K	11 zone (B, No A bit)
BWZ	S	Zero zone (A, No B bit)
BWZ	3	Either a word mark, or no zone
BWZ	C	Either a word mark, or 12 zone
BWZ	L	Either a word mark, or 11 zone
BWZ	T	Either a word mark, or zero zone

Figure 33. Branch if Word Mark and/or Zone Mnemonics, d-Characters, and Conditions

Branch If Word Mark and/or Zone

Instruction Format.

Mnemonic	*Op Code*	*I-address*	*B-address*	*d-character*
See Figure 33.	V	xxx	xxx	x

Function. This instruction examines the character located at the B-address for the zone or word-mark combinations specified by the d-character. A correct comparison branches the program to the specified I-address. If the program does not branch to the I-address, it continues with the next sequential instruction. The d-characters, the associated mnemonics, and the conditions they test are shown in Figure 33.

Word Marks. These have been explained previously.

Timing.

No Branch:

$T = .0111 (L_I + 2)$ ms.

Branch (without indexing):

$T = .0111 (L_I + 2)$ ms.

Branch (with indexing):

$T = .0111 (L_I + 3)$ ms.

Address Registers After Opertion.

	I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
No Branch	NSI	BI	B-1
Branch (without indexing)	NSI	BI	blank
Branch (with indexing)	NSI	BI	NSI

Example. Test the units position of GROAMT (2498) for an 11-zone, and branch to NEGRTE (0598) for the next instruction. If there is no 11-zone, continue the program sequence (Figure 34).

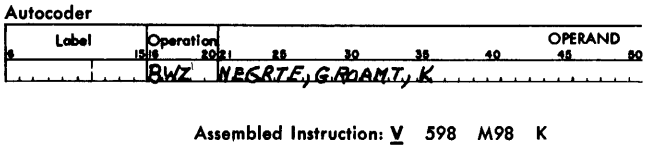


Figure 34. Branch if Word Mark and/or Zone

Compare

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*
C	C	xxx	xxx

Function. The characters in the A-field are compared to an equal number of characters in the B-field. The comparison turns on an indicator that can be tested by a subsequent BRANCH IF INDICATOR ON instruction. The indicator is reset by either the next 7-character COMPARE instruction or the next disk-storage operation.

The same indicators set by the COMPARE instruction are also affected by a disk operation (seek, read, write, and write check). The disk-storage drive performs an address-compare operation automatically on the address in core storage, with the address on the disk record, by using the compare circuits and by setting the appropriate indicator (equal, high, or low). Therefore, careful consideration must be made in the use of a COMPARE instruction and subsequent BRANCH IF INDICATOR ON instructions for testing the results of the COMPARE instruction when disk operations are to be performed.

Word Marks. The first word mark encountered stops the operation. If the A-field is longer than the B-field, extra A-field positions at the left of the B-field word mark are not compared. If the B-field is longer than the A-field, an unequal-compare results.

Timing. $T = .0111\ (L_I + 1 + 2L_W)$ ms.

Note. Both fields must have exactly the same bit configurations to be equal. For example, 00? (? = $\bar{0}$) compared to 00! (! = $\bar{0}$) results in an unequal comparison.

All characters that can appear in storage can be compared. The ascending sequence of characters is:

blank • ¤ [< ≢ & $ *] ; Δ — / , % ~ \ ≠ ♭ # @ : > √ ? A through I ! J through R ≠ S through Z 0 through 9.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-L_W	B-L_W

Example. Compare the department numbers punched in two cards. Department numbers are located in:

Card	*Label*	*Actual Address*
1	DEPTNO	1098
2	DEPTCD	0004

Then test the result of the compare operation. If the department numbers are equal, continue the program in sequence. If they are unequal, branch to TOTAL (0495) for the next instruction (Figure 35).

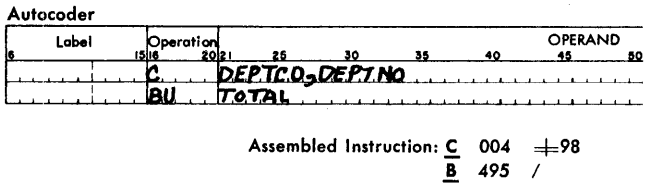


Figure 35. Compare

Data-Moving Operations

The 1440 data-moving operations are used to manipulate data within core storage during processing. Depending on the specific operation, one character, a group of characters, or a part of one character can be involved in the operation. A move operation does not affect word marks, but a load operation causes word marks, as well as data, to be transferred.

Data-Moving Instructions

Move Characters to A or B Word Mark (Two Fields)

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*
MLC	M	xxx	xxx

Function. The data in the A-field is moved to the B-field.

Word Marks. If both fields are the same length, only one of the fields must have a defining word mark. The first word mark encountered stops the operation. If the word mark is sensed in the A-field, the

machine takes one more B-cycle to move the high-order character from A to B. At the end of the operation, the A-address register and the B-address register contain the addresses of the storage locations immediately to the left of the A- and B-fields processed by the instruction. The data at the A-address is unaffected by the move operation. Word marks in both fields are undisturbed.

Timing. $T = .0111 (L_I + 1 + 2L_W)$ ms.

Note. If the fields are unequal in length, chaining can produce unwanted results, because one of the fields has not been completely processed. Thus, one of the registers will *not* contain the address of the units position of the left-adjacent field.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-L_W	B-L_W

Example. Move the 5-character field NAMIN (0750) to the 5-character field NAMOUT (0850), Figure 36.

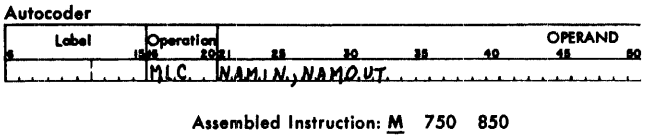


Figure 36. Move Characters to A or B Word Mark (Two Fields)

Move Characters to A or B Word Mark (One Field)

Instruction Format.

Mnemonic	*Op Code*	*A-address*
MLC	M	xxx

Function. This format of the move operation can be used when it is desired to move fields from the A-area and store them sequentially in the B-area. It saves program storage space and time, because the B-address is automatically taken from the B-address register, and does not have to be written or interpreted as part of the instruction.

Word Marks. A word mark is required in the high-order position of the A- or B-field. The first word mark encountered stops the move operation.

Timing. $T = .0111 (L_I + 1 + 2L_W)$ ms.

Note: If the B-address register already contains the correct address, the B-label of the first instruction in the example can be eliminated:

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-L_W	Bp-L_W

Example. Move the following three fields (labeled EMPNO, DEPTNO and TAXCLS) and store them sequentially at RECOUT (units position at 0204), Figure 37.

	A-label	*A-actual address*	*B-label*	*B-actual address*
Employee number	EMPYNO	0101-0104		0201-0204
Department	DEPTNO	0108-0110		0205-0207
Tax Class	TAXCLS	0114-0115	RECOUT	0208-0209

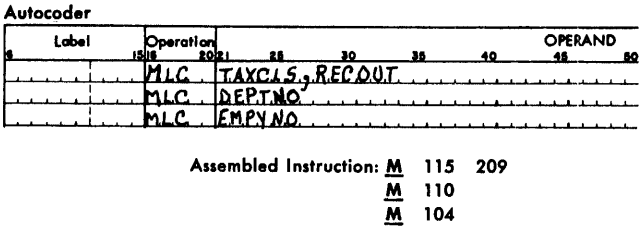


Figure 37. Move Characters to A or B Word Mark (One Field)

Move Characters and Suppress Zeros

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*
MCS	Z	xxx	xxx

Function. The data in the A-field is moved to the B-field. After the move, high-order zeros and commas are replaced by blanks in the B-field. Any character that is not a comma, hyphen, blank, significant digit, or zero causes zero suppression to begin again. The sign is removed from the units position of the data field. Refer to Figure 38 for a move characters and suppress zeros operation example.

Example	Op Code	A–address	B–address
Move Char. and Suppress Zeros	Z	xxx	xxx
Storage before		A–field (data)	B–field (data)
		± 001206	bbbbbbb
Storage after		± 001206	bbb1206

Figure 38. Move Characters and Suppress Zeros Operation Example

Example	Op Code	A–address	B–address
Move Char. and Suppress Zeros	Z	xxx	xxx
Storage before		A–field (data)	B–field (data)
		± 0010b @ 00.25	bbbbbbbbbbbb
Storage after		± 0010b @ 00.25	bbb10b @ bb.25

Figure 39. Move Characters and Suppress Zeros Operation Example, Multiple Field

Figure 39 is another example of a move characters and suppress zeros operation involving a multiple field transfer. In this operation there are effectively two groups of high-order zeros. The @ sign is recognized as not being a significant digit or a zero, blank, comma, decimal, or minus sign. Thus, not only are the two high-order zeros suppressed, but also the two zeros to the right of the @ sign.

Word Marks. The A-field word mark stops transmission of data. B-field word marks, encountered during the move operation, are erased.

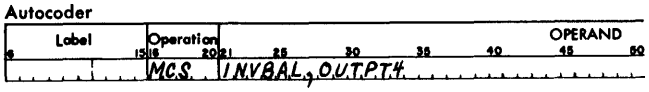
Timing. $T = .0111 (L_I + 1 + 3L_A)$ ms.

Note. This description of the instruction assumes a 1440 system without the expanded print edit special feature. If the feature is installed, a decimal does not restart zero suppression.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	$A\text{-}L_A$	B + 1

Example. Move and suppress the zeros in the 10-character field labeled INVBAL (0958) to the area labeled OUTPT4 (0448), Figure 40.



Assembled Instruction: Z 958 448

Figure 40. Move Characters and Suppress Zeros

Move Characters to Record Mark or Group-Mark with a Word-Mark

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*
MRCM	P	xxx	xxx

Function. This instruction makes it possible to move an entire record from one core-storage area to another, regardless of the presence of word marks in either field. The A- and B-addresses specify the high-order position of the respective areas. Transmission starts from the high-order addresses, and continues until a record mark (A82 bits) or a group-mark with a word-mark (WMBA8421 bits) is sensed in the A-field. The record mark or group mark transfers to the B-field.

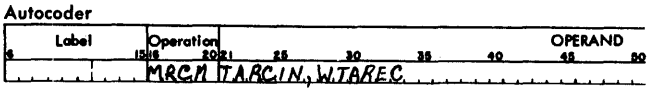
Word Marks. Word marks within the area do not affect the operation. Any word marks in the B-field remain unchanged. A-field word marks are not transmitted to the B-field.

Timing. $T = .0111 (L_I + 1 + 2L_A)$ ms.

Address Registers After Operation

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	$A + L_A$ (The length of the A-field includes the group-mark with a word-mark or record mark)	$B + L_A$

Example. Move the disk record that has its high-order character in the location labeled DARCIN (0679) to another area of core storage beginning at the label WDAREC (0985), Figure 41.



Assembled Instruction: P 679 985

Figure 41. Move Characters to Record Mark or Group-Mark with a Word-Mark

Move Numerical

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*
MLNS	D	xxx	xxx

Function. The numerical portion (8-4-2-1 bits) of the single character in the A-address is moved to the B-address. The zone portions (AB bits) are undisturbed at both addresses. The entire character in the A-address is left undisturbed.

Word Marks. Word marks are not required at either address, because the nature of the instruction always specifies that only one digit is to be transmitted.

Timing. $T = .0111 (L_I + 3)$ ms.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-1	B-1

Example. Move the numerical portion of the units position of ONHAND (0986) to OUT5 (0789), Figure 42.

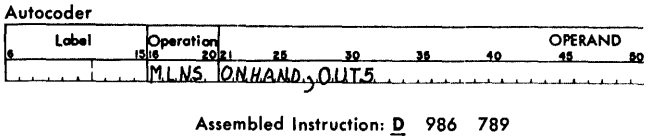


Figure 42. Move Numerical

Move Zone

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*
MLZS	Y	xxx	xxx

Function. Only the zone portion (AB bits) is moved from the A-address to the B-address. The digit portions (8-4-2-1 bits) are undisturbed at both addresses. The entire character in the A-address is left undisturbed.

Word Marks. Word marks are not required at either the A- or B-addresses, because this instruction involves a single character.

Timing. $T = .0111 (L_I + 3)$ ms.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-1	B-1

Example. Move the zone bits from the units position of NEWBAL (3100) to the area labeled REC2 (3195), Figure 43.

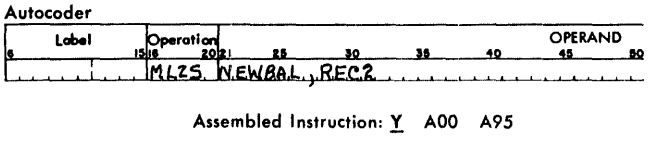


Figure 43. Move Zone

Load Characters to A Word Mark (Two Fields)

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*
MLCWA	L	xxx	xxx

Function. This instruction is commonly used to load data into designated printer or punch output areas of storage, and also to transfer data or instructions from a designated read-in area to another storage area. The data and word mark from the A-field are transferred to the B-field, and all other word marks in the B-field are cleared.

Word Marks. The A-field must have a defining word mark, because the A-field word mark stops the operation.

Timing. $T = .0111 (L_I + 1 + 2L_A)$ ms.

Note: If the B-field is larger than the A-field, the B-field word mark is not cleared.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	$A\text{-}L_A$	$B\text{-}L_A$

Example. Transfer the data and word marks from REC4 (0950) to OUT8 (0650), Figure 44.

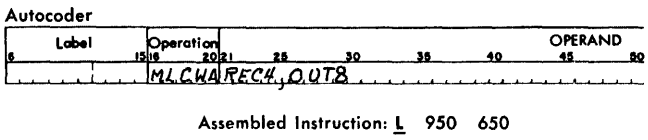


Figure 44. Load Characters to A Word Mark (Two Fields)

Load Characters to A Word Mark (One Field)

Instruction Format.

Mnemonic	*Op Code*	*A-address*
MLCWA	L	xxx

Function. This format can be used when several A-fields (not necessarily in sequence) are to be loaded sequentially in the B-field. This instruction causes the A-field data and word mark to be moved to the B-field. B-field word marks are cleared, up to the A-field word mark.

Word Marks. The A-field word mark stops the operation. Therefore, B-field word marks, beyond the left limit of the A-field, are not cleared.

Timing. $T = .0111 (L_I + 1 + 2L_A)$ ms.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	$A\text{-}L_A$	$Bp\text{-}L_A$

Example. Load the three fields, EMPYNO, DEPTNO, and TAXCLS, with their word marks to sequential locations, beginning at storage location (0201), Figure 45.

	A-label	*A-actual address*	*B-label*	*B-actual address*
Employee number	EMPYNO	0101-0104		0201-0204
Department	DEPTNO	0108-0110		0205-0207
Tax Class	TAXCLS	0114-0115	PRINT1	0208-0209

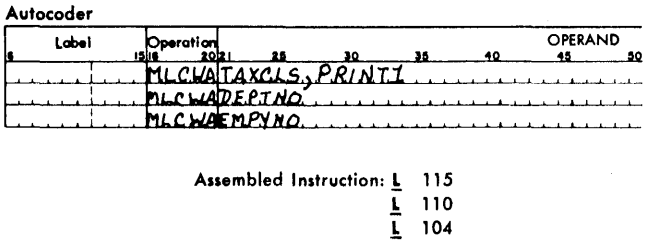


Figure 45. Load Character to A Word Mark (One Field)

Miscellaneous Operations

The miscellaneous operations in an IBM 1440 Data Processing System involve the insertion and removal of word marks from specific core-storage locations, the clearing of core-storage areas, programmed halt operations, and other similar operations.

Miscellaneous Instructions

Set Word Mark (Two Addresses)

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*
SW	;	xxx	xxx

Function. A word mark is set at each address specified in the instruction. The data at each address is undisturbed. A word mark cannot be set in core-storage position 000.

Word Marks. Word marks are set at both the A- and B-addresses specified. A word mark is not required in the core-storage position adjacent to this instruction.

Timing. $T = .0111 (L_I + 2)$ ms.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-1	B-1

Example. Set word marks at locations BEGIN1 (3950) and BEGIN2 (3970), Figure 46.

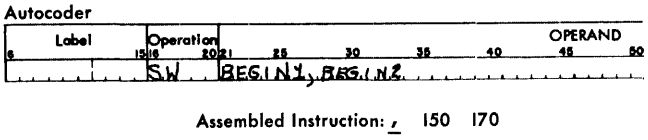


Figure 46. Set Word Mark (Two Addresses)

Set Word Mark (One Address)

Instruction Format.

Mnemonic	*Op Code*	*A-address*
SW	,	xxx

Function. This format of the SET WORD MARK instruction causes a word mark to be set at the A-address. Data at this address is undisturbed. A word mark cannot be set in core-storage position 000.

Word Marks. A word mark is set at the A-address.

Timing. $T = .0111 (L_I + 3)$ ms.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-1	A-1

Example. Set a word mark at AREA2 (2901), Figure 47.

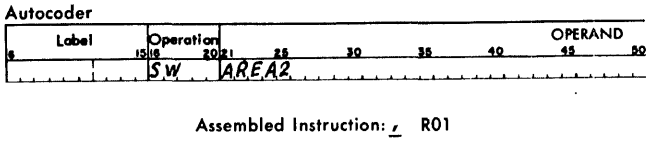


Figure 47. Set Word Mark (One Address)

Clear Word Mark (Two Addresses)

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*
CW	□	xxx	xxx

Function. This instruction clears word marks at the locations specified by the A- and B-addresses, without disturbing the data there. A process error occurs if the specified A- or B-address is core-storage position 000 (end-around check condition).

Word Marks. Word marks are cleared at the A- and B-addresses.

Timing. $T = .0111 (L_I + 3)$ ms.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-1	B-1

Example. Clear the word marks at NETPAY (1924) and ACCUM4 (3309), Figure 48.

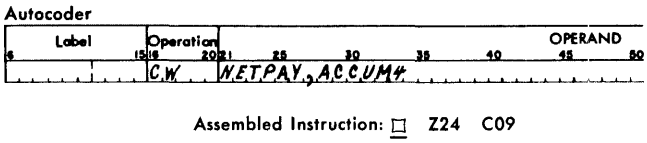


Figure 48. Clear Word Mark (Two Addresses)

Clear Word Mark (One Address)

Instruction Format.

Mnemonic	*Op Code*	*A-address*
CW	⊐	xxx

Function. This format of the CLEAR WORD MARK instruction causes the word mark to be cleared at the A-address. Data at the A-address is not disturbed. A process error occurs if the specified A-address is core-storage position 000 (end-around check condition).

Word Marks. Word marks are cleared at the A-address only.

Timing. $T = .0111 (L_I + 3)$ ms.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A-1	A-1

Example. Clear the word mark at RECNO1 (3608), Figure 49.

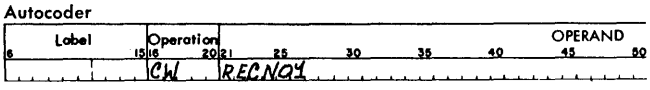


Figure 49. Clear Word Mark (One Address)

Clear Storage

Instruction Format.

Mnemonic	*Op Code*	*A-address*
CS	/	xxx

Function. As many as 100 positions of core storage can be cleared of data and word marks when this instruction is executed. Clearing starts at the A-address and continues in descending address sequence to the nearest hundreds position. The cleared area is set to blanks.

Word Marks. Word marks do not stop the operation.

Timing. $T = .0111 (L_I + 1 + L_X)$ ms.

Note: During the execution of this instruction, only the B-address register is used. Therefore, when chaining is being considered, the contents of the A-address register can be ignored.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A	x 00-1

Example. Clear WAREA5 (0500-0563), Figure 50.

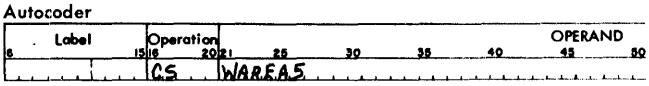


Figure 50. Clear Storage

Clear Storage and Branch

Instruction Format.

Mnemonic	*Op Code*	*I-address*	*B-address*
CS	/	xxx	xxx

Function. This is the same as the CLEAR STORAGE instruction, except that the clearing starts at the B-address. The I-address specifies the location of the next instruction.

Word Marks. Word marks do not stop the operation. It is not necessary to follow this instruction with a character and an associated word mark.

Timing.

Without indexing:

$T = .0111 (L_I + L_X)$ ms.

With indexing:

$T = .0111 (L_I + 1 + L_X)$ ms.

Address Registers After Operation.

	I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
Without indexing	NSI	BI	blank
With indexing	NSI	BI	NSI

Example. Clear WAREA2 (0800-0898) and branch to START4 (0498) for the next instruction (Figure 51).

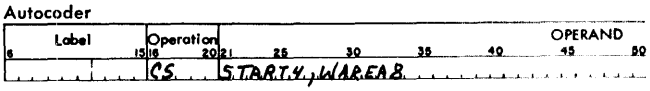


Figure 51. Clear Storage and Branch

No Operation

Instruction Format.

Mnemonic	*Op Code*
NOP	N

Function. This code performs no operation. It can be substituted for the operation code of any instruction to make that instruction ineffective. It is commonly used in program modification to cause the machine to skip over specific instructions.

Instructions that have A-addresses of %xx or @xx should have their A-address field set to valid numeric values (all zeros, for example), or all N's with associated word marks to perform a no-operation function successfully. If this is not done, the A-address may contain characters that cause indexing and/or invalid core-storage addressing problems.

Word Marks. The program operation resumes at the next operation code identified by a word mark.

Timing. $T = .0111 (L_I + 1)$ ms.

Note. If characters without word marks follow an N operation code, these characters enter the A- and B-field registers. For example:

N 1234 A xxxx

In this instance, the address registers after operation would be:

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	123	4bb*

* If this address is subsequently used (chained or stored) an invalid-address check stop condition occurs.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	A	B

Example. Leave eight storage positions open for an instruction code such as READ CARD M (000) (000) R. The correct instruction can be inserted when needed (Figure 52).

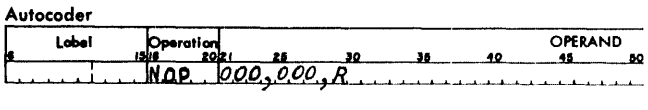


Figure 52. No Operation

Halt

Instruction Format.

Mnemonic	*Op Code*
H	.

Function. This instruction causes the machine to stop and the stop-key light to turn ON. Pressing the start key causes the program to start at the next instruction in sequence.

Word Marks. Word marks are not affected.

Timing. $T = .0111 (L_I + 1)$ ms.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	Ap	Bp

Example. Figure 53 is a symbolic example of the HALT instruction.

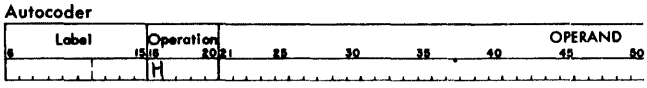


Figure 53. Halt

Halt and Branch

Instruction Format.

Mnemonic	*Op Code*	*I-address*
H	.	xxx

Function. This is the same as HALT, except that the next instruction is at the I-address.

Word Marks. Word marks are not affected.

Timing.

Without indexing:
$T = .0111 (L_I + 1)$ ms.

With indexing:
$T = .0111 (L_I + 2)$ ms.

Address Registers After Operation.

	I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
Without indexing	NSI	BI	blank
With indexing	NSI	BI	NSI

Example. Stop the system, and branch to START2 (0895) for the next instruction when the start key is pressed (Figure 54).

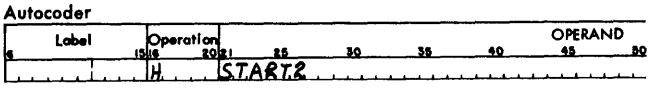


Figure 54. Halt and Branch

Coded Halt

Instruction Format.

Mnemonic	*Op Code*	*I-address*	*B-address*	*d-character*
H	±	C_1		
	±	C_1 C_2		
	±	C_1 C_2 C_3	C_4	
	±	C_1 C_2 C_3	C_4 C_5	
	±	C_1 C_2 C_3	C_4 C_5 C_6	
	±	C_1 C_2 C_3	C_4 C_5 C_6	C_7

Function. These forms of the HALT instruction place coded information in the A- and B-address and d-character positions. The coded information is then used to identify the halt. The coding used in these positions is left to the discretion of the programmer, but the system's valid addressing and indexing rules must be followed. For example, the first four forms of this instruction, as listed in the instruction format, leave invalid addresses in the A- and/or B-address registers and these addresses cannot be used in subsequent operations.

Word Marks. A word mark is required in the core-storage position adjacent to the instruction to specify the instruction length.

Timing. $T = .0111\ (L_I + 1)$ ms.

Note. The last coded character also appears in the A-register.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	C_1 b b	C_1 b b
NSI	C_1 C_2 b	C_1 C_2 b
NSI	C_1 C_2 C_3	C_4 b b
NSI	C_1 C_2 C_3	C_4 C_5 b
NSI	C_1 C_2 C_3	C_4 C_5 C_6
NSI	C_1 C_2 C_3	C_4 C_5 C_6

Example. Stop the system, and label the stop as 22, (Figure 55).

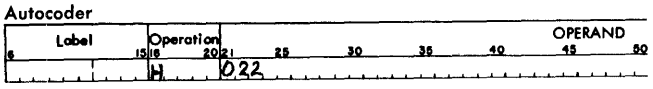


Figure 55. Coded Halt

Edit Operation

The IBM 1440 Data Processing System has a powerful edit instruction that can cause all desired commas, decimals, dollar signs, asterisks, credit symbols, and minus signs to be inserted automatically in a numerical output field. Unwanted zeros to the left of significant digits can be suppressed. Thus, editing in the 1440 system is the automatic control of zero suppression, inserting of identifying symbols, and punctuation of an output field (Figure 56).

In editing, two fields are needed — the data field and a control field. The data field is the data edited for output. The control field specifies how the data field is edited. It specifies the location of punctuation and condition of special characters and indicates where zero suppression occurs. The two fields are operated on character-by-character, under control of editing rules.

The control word has two parts: the *body* (which punctuates the A-field), and the *status* portion (which contains the dollar signs, sign-symbols, and class of total asterisks). The sign of the A-field determines whether or not sign symbols will print. The sign of the A-field is removed.

To edit a field, a LOAD CHARACTERS TO A WORD MARK instruction loads the control word into the specified printer output area. This puts the control word where the edited information will eventually go. Then, a MOVE CHARACTERS AND EDIT instruction (with the same B-address as the previous load instruction) performs the editing function as it moves the data into the output area.

NOTE: A 1-position field cannot be edited. Figure 57 shows the use of these rules as applied to the data in Figure 56.

Move Characters and Edit

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*
MCE	E	xxx	xxx

Function. The data field (A-field) is modified by the contents of the edit control field (B-field), and the result is stored in the B-field. The data field and the control field are read from storage character-by-character, under control of the word marks and the editing rules. Any sign in the units position of the data field is removed during the operation.

	OP	A-address	B-address
Edit instruction	E	789	300
		A-field (data)	B-field (control word)
Storage		00257426	$ bbb, bb0.bb & CR &**
			B-field
Result of edit		00257426	$ 2,574.26 **

Figure 56. Editing Operation

EDITING RULES

Rule 1. All numerical, alphabetic, and special characters can be used in the control word. However, some of these characters have special meanings:

Control Character	*Function*
b (blank)	This is replaced with the character from the corresponding position of the A-field.
0 (zero)	This is used for zero suppression, and is replaced with a corresponding character from the A-field. Also the right-most "0" in the control word indicates the right-most limit of zero suppression.
. (decimal)	This remains in the edited field in the position where written. It is removed during a zero-suppress operation if it is to the left of the high-order significant digit. When used with the expanded print edit feature, it has an additional function (see *Expanded Print Edit* section, *Special Features,* Form A26-5669).
, (comma)	This remains in the edited field in the position where written. It is removed during a zero-suppress operation if it is to the left of the high-order significant digit.
CR (credit)	This is undisturbed if the data sign is negative. It is blanked out if the data sign is positive. It can be used in body of control word without being subject to sign control.
— (minus)	This is the same as CR.
& (ampersand)	This causes a space in the edited field. It can be used in multiples.
* (asterisk)	This can be used in singular or in multiples, usually to indicate class of total. When it is used with the expanded print edit feature, it takes on an additional function (see *Expanded Print Edit* section, *Special Features,* Form A26-5669).
$ dollar sign	This is undisturbed in the position where it is written. When used with the expanded print edit feature, it has an additional function (see *Expanded Print Edit* section, *Special Features,* Form A26-5669).

Rule 2. A word mark in the high-order position of the B-field controls the move characters and edit operation.

Rule 3. When the A-field word mark is sensed, the remaining commas in the control field are set to blanks.

Rule 4. The body of the control word is that portion beginning with the right-most blank or zero, and continuing to the left to the control character that governs the transfer of the last position of the data field. The remaining portion of the control field is the *status* portion.

Cycle	TYPE OF CYCLE	ADDRESS REGISTERS			REG.		PUT BACK INTO STORAGE	"B" FIELD AT END OF CYCLE	REMARKS
		I	A	B	B	A			
1	I_{op}	002	?	?	E		E	$bbb,bb0.bb&CR&**	Read Instr. OP Code
2	I_1	003	07bb	07bb	7	7	7	same	Load A Address Register
3	I_2	004	078b	078b	8	8	8	same	Load A Address Register
4	I_3	005	0789	0789	9	9	9	same	Load A Address Register
5	I_4	006	0789	03bb	3	3	3	same	Load B Address Register
6	I_5	007	0789	030b	0	0	0	same	Load B Address Register
7	I_6	008	0789	0300	0	0	0	same	Load B Address Register
8	I_7	008	0789	0300	OP	0	OP	same	OP code of next instr.
9	A	008	0788	0300	6	6	6	same	Execute EDIT instr.
10	B	008	0788	0299	*	6	*	same	Rule 1
11	B	008	0788	0298	*	6	*	same	Rule 1
12	B	008	0788	0297	&	6	Blank	$bbb,bb0.bb&CRb**	Rule 1
13	B	008	0788	0296	R	6	Blank	$bbb,bb0.bb&Cbb**	Rule 1 and 5
14	B	008	0788	0295	C	6	Blank	$bbb,bb0.bb&bbb**	Rule 1 and 5
15	B	008	0788	0294	&	6	Blank	$bbb,bb0.bbbbbb**	Rule 1
16	B	008	0788	0293	b	6	6	$bbb,bb0.b6bbbb**	Rule 1
17	A	008	0787	0293	2	2	2	same	Rule 1
18	B	008	0787	0292	b	2	2	$bbb,bb0.26bbbb**	Rule 1
19	A	008	0786	0292	4	4	4	same	Rule 1
20	B	008	0786	0291	.	4	.	same	Rule 1
21	B	008	0786	0290	0	4	4	$bbb,bb4.26bbbb**	Zero Suppress—Rule1 and 6
22	A	008	0785	0290	7	7	7	same	Rule 1
23	B	008	0785	0289	b	7	7	$bbb,b74.26bbbb**	Rule 1
24	A	008	0784	0289	5	5	5	same	Rule 1
25	B	008	0784	0288	b	5	5	$bbb,574.2bbbbb**	Rule 1
26	A	008	0783	0288	2	2	2	same	Rule 1
27	B	008	0783	0287	,	2	,	same	Rule 1
28	B	008	0783	0286	b	2	2	$bb2,574.26bbbb**	Rule 1
29	A	008	0782	0286	0	0	0	same	Rule 1
30	B	008	0782	0285	b	0	0	$b02,574.26bbbb**	Rule 1
31	A	008	0781	0285	0	0	0	same	Rule 1
32	B	008	0781	0284	b	0	0	$002,574.26bbbb**	Rule 1
33	B	008	0781	0284	$	0	$	$002,574.26bbbb**	Sense Word Mark—Rev. Scan—Rule 1 and 6
34	B	008	0781	0285	$	0	$	same	Rule 6
35	B	008	0781	0286	0	0	Blank	$b02,574.26bbbb**	Rule 6
36	B	008	0781	0287	0	0	Blank	$bb2,574.26bbbb**	Rule 6
37	B	008	0781	0288	2	0	2	same	Rule 6
38	B	008	0781	0289	,	0	,	same	Rule 6
39	B	008	0781	0290	5	0	5	same	Rule 6
40	B	008	0781	0291	7	0	7	same	Rule 6
41	B	008	0781	0292	4	0	4	$bb2,574.26bbbb**	Rule 6

Figure 57. Step-by-Step Editing Operation

Rule 5. If the data field is positive, and if the CR or — symbols are located in the status portion of the control word, they are blanked out.

Rule 6. The data field can contain fewer, but must not contain more positions than the number of blanks and zeros in the body of the control word. Dollar signs and asterisks are included in the body of the control word with the expanded print edit special feature.

Rule 7. Zero suppression is used if unwanted zeros to the left of significant digits in a data field are to be deleted (see Figure 58).

ZERO SUPPRESSION OPERATION

Zero suppression is the deletion of unwanted zeros at the left of significant digits in an output field (Figure 58).

A special 0 is placed (in the body of the control word) in the right-most limit of zero suppression.

To perform zero-suppression operations properly, there must be at least one character to the left of the zero-suppression character in the control word.

Forward Scan:

1. The positions in the output field at the right of this special zero are replaced by the corresponding digits from the A-field.

A-field	0010900
Control word (B-field)	$ bb, bb0. bb
Forward scan	$ 00,109. 00
Reverse scan	$ bbb109. 00
Results of edit	$ 109. 00

Figure 58. Zero Suppression Operation

2. The special zero is replaced by the corresponding digit from the A-field, when it is detected in the control field.
3. A word mark is automatically set in this position of the B- (output) field.
4. The scan continues until the B-field (high order) word mark is sensed and removed.

Reverse Scan:

1. In the output field, blanks replace all zeros and punctuation, except hyphens at the left of the first significant character (up to, and including, the zero-suppression code position).
2. When the automatically-set zero suppression word mark is sensed, it is erased and the operation ends.

Timing. $T = .0111\ (L_I + 1 + L_A + L_B + L_Y)$ ms.

Address Registers After Operation.

	I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
Without zero suppression	NSI	$A\text{-}L_A$	$B\text{-}L_B$
With zero suppression	NSI	$A\text{-}L_A$	Location of the special control zero plus 1.

Example. Edit the data labeled GROPAY (0985) by the edit-control word EDCONT (0325). Store the result in PRINT6 (00250), Figure 59.

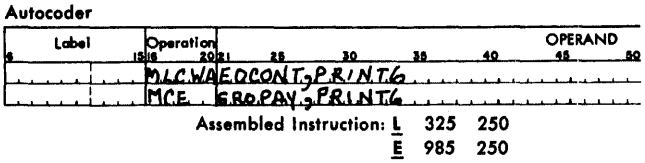


Figure 59. Move Characters and Edit

IBM 1447 Console Operations

The IBM 1447 Console (Model 1, 2, or 4), Figure 60, is a required unit on an IBM 1440 Data Processing System. The console contains the system operating keys, lights and switches which give the operator external control for setting up and checking system operation. For more detail on the keys, lights, switches, and operating procedures, refer to IBM *1447 Console,* Form A24-3031.

Console Instruction Format

A program-initiated data transmission between the IBM 1447 Console (Model 2 or 4) and the attached system is started by executing the proper console instruction. If the data transmission is from the 1447 console to the system, a READ FROM 1447 CONSOLE instruction is executed. The format for the 1447 console is shown in Figure 61.

The various parts of a 1447 console instruction and their uses are:

General Mode of Operation

This part of the instruction identifies the operation as either a move operation or a load operation. A move operation specifies that only the character coding is transmitted. A load operation specifies that both the character coding and any associated word marks are transmitted.

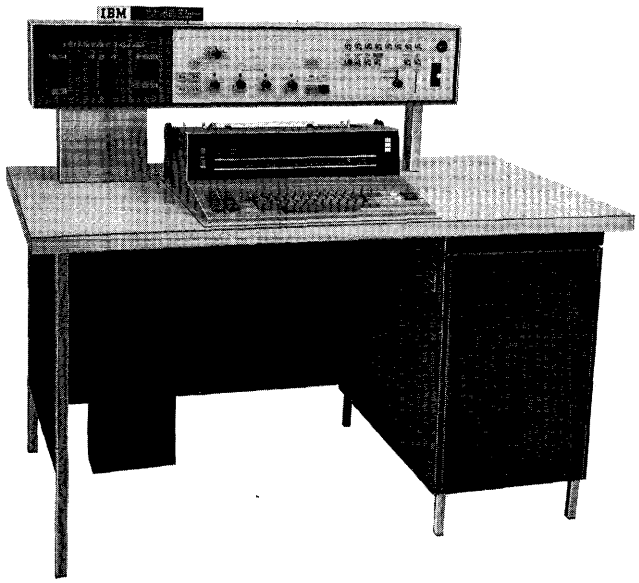


Figure 60. IBM 1447 Console, Model 2

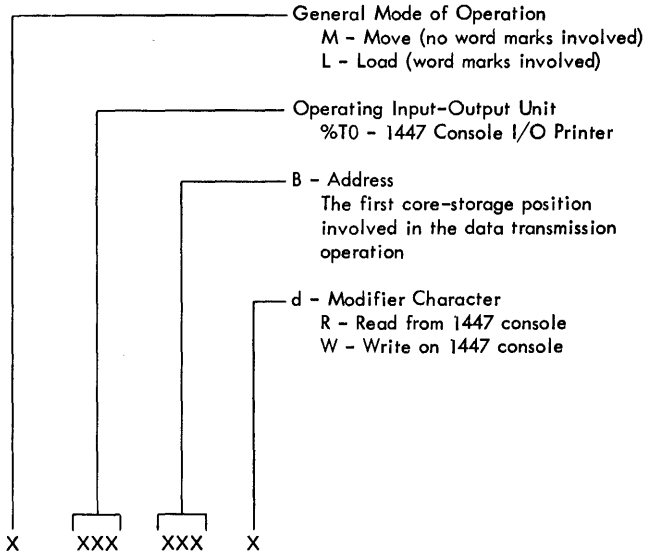


Figure 61. 1447 Console I/O Printer Instruction Format

Operating Input/Output Unit

This part of the instruction specifies the console I/O printer as the active input/output unit for this operation.

B-Address

This part of the instruction specifies the first core-storage position that will be involved in the operation.

d-Modifier Character

This part of the instruction specifies the data transmission direction. An R specifies a console printer-to-system data transmission; a W specifies a system-to-console printer data transmission.

IBM 1447 Console Instructions

Read from 1447 Console

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*	*d-character*
RCP	M/L	%T0	BBB	R

Function. This instruction is used to enter data into core storage from the console I/O printer. The Op code specifies the mode of operation. If the operation takes place in the *move* mode (M Op code), word marks cannot be transmitted from the console printer into core storage. Any word marks already in the area that accepts the message will remain there.

If the operation takes place in the *load* mode (L Op code), word marks can be transmitted from the console printer into core storage when the word-mark key is pressed. Any word marks already in the area that accepts the message will be removed.

The A-address specifies the console I/O printer as the I/O unit involved in the operation. The B-address specifies the first core-storage position that accepts data from the console printer. The d-character specifies a console printer-to-system operation.

The console operator can start keying the data when the white type light on the console comes ON. The console operator prints the data on the console printer and the characters enter core storage, beginning at the location specified by the B-address portion of the instruction.

The operator transmits a word mark by pressing the shift key and the word-mark key. The upper case (word-mark position) of the period key prints an inverted circumflex. The next character printed will enter a core-storage position and have a word mark associated with it.

When the number of data positions to be entered into core storage exceeds the number of printing positions on one printer line, the print element automatically returns from the right-hand margin, executes a line feed in operation, and the keying operation continues on the next line.

The operation is normally ended when the operator presses the release key. This key operation inserts a group-mark with a word-mark in core storage, initiates a carrier-return and line-feed operation, and disconnects the printer from the system.

The operation can also be ended if a group-mark with a word-mark is sensed in core storage. This signifies that the input message exceeded the core-storage area capacity and:

1. The operation ends and the printer is disconnected from the system.
2. The inquiry clear (*) indicator in the system comes on.
3. The red type light on the console comes on.
4. A carrier-return and line-feed operation is initiated.
5. The keyboard locks up.

Word Marks. Depends on mode of operation. To end the operation correctly, a group-mark with a word-mark must be inserted into the 1440 core-storage position to the right of the position that contains the last character sent to the system from the console printer.

Timing. T = .0111 (L_I + 1) ms + operator keying time.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	%30	B + L_B + 1

Example. Transfer the data keyed on the console I/O printer to the area in 1440 core storage labeled INQIN (0785), Figure 62.

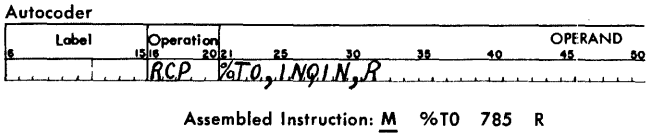


Figure 62. Read from 1447 Console

Write on 1447 Console

Instruction Format.

Mnemonic	*Op Code*	*A-address*	*B-address*	*d-character*
WCP	M/L	%T0	BBB	W

Function. This instruction is used to transfer data from core storage to the console I/O printer. The Op code specifies the mode of operation. If the operation takes place in the move mode, word marks are ignored. The character with an associated word mark in core storage is printed as a character only. Functional control characters cause the specified carrier movement on the console printer, and the characters do not print. Refer to IBM *1447 Console* (Form A24-3031) for functional control characters and associated printer operation.

If the operation takes place in the load mode, the word marks are transmitted and printed. The word mark is printed before the associated character is printed. Functional control characters are also printed. The carrier movement normally specified by the character does not occur.

The A-address specifies the console I/O printer as the I/O unit involved in the operation and turns on the white type light if the printer is available for use. The B-address specifies the first core-storage position of the area that contains the data to be printed. The d-character specifies a system-to-console printer operation.

The data reads out of core storage, beginning at the address specified in the instruction and continuing until a group-mark with a word-mark is en-

countered. The group-mark with a word-mark ends the operation, but does not print. A carrier-return operation, with an associated line-feed operation, occurs and the system advances to the next instruction.

If the end of a printed line is reached before the group-mark with a word-mark is sensed, printing is suspended and a carrier-return and line-feed operation is executed. When the carrier reaches the left-hand margin, the print-out operation continues.

Word Marks. Depends on mode of operation. A group-mark with a word-mark in core storage ends the operation.

Timing. $T = .0111 (L_I + 1) + 68 (L_B) + 800$ (number of carrier return operations -1) ms.

Address Registers After Operation.

I-Add. Reg.	*A-Add. Reg.*	*B-Add. Reg.*
NSI	%30	$B + L_B + 1$

Example. Print out the data, beginning in the area labeled INQOUT (0785) and ending with a group-mark with a word-mark (Figure 63).

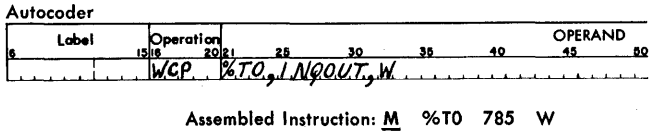


Figure 63. Write on 1447 Console

Console I/O Printer Timing

The console I/O printer is used for input to, and output from, the IBM 1440 Data Processing System.

The timing involved during an input operation is: $T = .0111 (L_I + 1)$ + console operator keying time.

The timing involved during an output operation is: $T = .0111 (L_I + 1) + 68 (L_B) + 800$ (number of carrier return operations -1) ms.*

* All system-console printer operations are unbuffered operations. Only one portion of either operation is overlapped by processing. This is the last carrier-return and line-feed operation that occurs at the end of an output operation.

Appendix

DEFINED CHARACTER	CARD CODE	BCD	CODE	13	39	52 A	52 H	63	DEFINED CHARACTER	CARD CODE	BCD	CODE	13	39	52 A	52 H	63
Blank		C		X	X	X	X	X	G	12-7	B A	4 2 1		X	X	X	X
Period	12-3-8	B A 8	2 1	X	X	X	X	X	H	12-8	B A 8			X	X	X	X
¤ Lozenge	12-4-8	C B A 8 4				¤	)	X	I	12-9	C B A 8	1		X	X	X	X
[Left Bracket	12-5-8	B A 8 4	1					X	! (- zero)	11-0	B	8 2			X	X	X
< Less Than	12-6-8	B A 8 4	2					X	J	11-1	C B	1		X	X	X	X
‡ Group Mark	12-7-8	C B A 8 4	2 1					X	K	11-2	C B	2		X	X	X	X
& Ampersand	12	C B A				&	+	X	L	11-3	B	2 1		X	X	X	X
$ Dollar Sign	11-3-8	C B	8 2 1		X	X	X	X	M	11-4	C B	4		X	X	X	X
* Asterisk	11-4-8	B	8 4	X		X	X	X	N	11-5	B	4 1		X	X	X	X
] Right Bracket	11-5-8	C B	8 4 1					X	O	11-6	B	4 2		X	X	X	X
; Semicolon	11-6-8	C B	8 4 2					X	P	11-7	C B	4 2 1		X	X	X	X
△ Delta	11-7-8	B	8 4 2 1					X	Q	11-8	C B	8		X	X	X	X
- Hyphen	11	B		X		X	X	X	R	11-9	B	8 1		X	X	X	X
/ Diagonal	0-1	C A	1			X	X	X	≠ Record Mark	0-2-8	A	8 2			X	X	X
, Comma	0-3-8	C A 8	2 1		X	X	X	X	S	0-2	C A	2		X	X	X	X
% Percent Mark	0-4-8	A 8 4				%	(	X	T	0-3	A	2 1		X	X	X	X
∨ Word Separator	0-5-8	C A 8 4	1					X	U	0-4	C A	4		X	X	X	X
\ Left Oblique	0-6-8	C A 8 4	2					X	V	0-5	A	4 1		X	X	X	X
## Segment Mark	0-7-8	A 8 4	2 1					X	W	0-6	A	4 2		X	X	X	X
♭ Substitute Blank	2-8	A				X	X	X	X	0-7	C A	4 2 1		X	X	X	X
# Number Sign	3-8	8	2 1			#	=	X	Y	0-8	C A	8		X	X	X	X
@ At Sign	4-8	C	8 4			@	'	X	Z	0-9	A	8 1		X	X	X	X
: Colon	5-8	8 4	1			X	X	X	0 (Zero)	0	C	8 2	X	X	X	X	X
> Greater Than	6-8	8 4	2					X	1	1		1	X	X	X	X	X
√ Radical	7-8	C	8 4 2 1					X	2	2		2	X	X	X	X	X
? (Plus Zero)	12-0	C B A 8	2			X	X	X	3	3	C	2 1	X	X	X	X	X
A	12-1	B A	1		X	X	X	X	4	4		4	X	X	X	X	X
B	12-2	B A	2		X	X	X	X	5	5	C	4 1	X	X	X	X	X
C	12-3	C B A	2 1		X	X	X	X	6	6	C	4 2	X	X	X	X	X
D	12-4	B A	4		X	X	X	X	7	7		4 2 1	X	X	X	X	X
E	12-5	C B A	4 1		X	X	X	X	8	8		8	X	X	X	X	X
F	12-6	C B A	4 2		X	X	X	X	9	9	C	8 1	X	X	X	X	X

Figure 64. 1440 Character Code Chart in Collating Sequence

Index of 1440 Instructions

Instruction	Autocoder Mnemonic	Form	Page
Load Characters to A Word Mark (One Field)	MLCWA	L(A)	30
Load Characters to A Word Mark (Two Fields)	MLCWA	L(A)(B)	30
Modify Address (One Address)	MA	#(A)	17
Modify Address (Two Addresses)	MA	#(A)(B)	17
Move Characters and Edit	MCE	E(A)(B)	36
Move Characters and Suppress Zeros	MCS	Z(A)(B)	28
Move Characters to A or B Word Mark (One Field)	MLC	M(A)	28
Move Characters to A or B Word Mark (Two Fields)	MLC	M(A)(B)	27
Move Characters to Record Mark or Group-Mark with a Word-Mark	MRCM	P(A)(B)	29
Move Numerical	MLNS	D(A)(B)	29
Move Zone	MLZS	Y(A)(B)	30
No Operation	NOP	N	34
Read from 1447 Console	RCP	M/L(%T0)(B)R	39
Set Word Mark (One Address)	SW	,(A)	32
Set Word Mark (Two Addresses)	SW	,(A)(B)	32
Subtract (One Field)	S	S(A)	21
Subtract (Two Fields)	S	S(A)(B)	21
Write on 1447 Console	WCP	M/L(%T0)(B)W	40
Zero and Add (One Field)	ZA	?(A)	22
Zero and Add (Two Fields)	ZA	?(A)(B)	21
Zero and Subtract (One Field)	ZS	!(A)	23
Zero and Subtract (Two Fields)	ZS	!(A)(B)	22

Index

640120MHP

IBM 1440 Printed in U. S. A. A24-3116-0

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