HONEYWELL

HARDWARE

SERIES 200

PROGRAMMERS' REFERENCE MANUAL MODELS 200/1200 1250/2200/4200

SUBJECT: The Central Processor Hardware of Series 200 Models 200, 1200, 1250, 2200, and 4200; The Easycoder Assembly Language; Summary Information concerning Programming Series 200 Peripheral Devices and the Scientific Unit.

SPECIAL INSTRUCTIONS: This edition completely supersedes revision 1 of the Series 200 Programmers' Reference Manual (Models 200/1200/2200/4200), Order No. 139, dated November 10, 1966, and incorporates the information published in Addenda #1 and #2 to that manual. The portions of this publication containing new and changed information are indicated on page iii.

This volume and the manuals and bulletins pertaining to the peripheral components of an installed Series 200 system together constitute a programmers' handbook for that system.

DATE: October 1, 1968

FILE NO.: 113.0005.0000.2-139*

0502
5M
7.5969
Printed in U.S.A.

*Underscoring denotes Order Number.

PREFACE

This manual constitutes for the programmer a reference source of detailed information concerning the central processor hardware of Series 200 Models 200, 1200, 1250, 2200, and 4200. The Easycoder Assembly Language, used with the Series 200/Basic Programming System and the Operating System — Mod 1, is also defined. In addition, this volume contains summary information concerning the programming of Series 200 peripheral devices and the scientific unit. The hardware information presented herein is equally applicable for the programmer using the Series 200/Operating System — Mod 2. However, for this usage it should be supplemented by the information contained in Appendix C of the software manual Assembler J (Order No. 432).

Separate hardware manuals and bulletins contain detailed information about programming and operating individual Series 200 peripheral devices. Specific peripheral device publications are named in the tables of input/output control characters beginning on page 8-120 of this manual.

The only prerequisite for a thorough understanding of the information presented herein is a familiarity with basic data processing terminology. No previous knowledge of the Series 200 is assumed.

A programmers' handbook may be constructed by combining in a single binder this volume and the manuals/bulletins pertaining to the peripheral components of the installed Series 200 system. This manual and the peripheral device manuals are all published in loose-leaf format for ease of rapid updating by means of replacement-page addenda.

The equipment characteristics reported herein remain subject to change to allow the introduction of design improvements.

Wellesley Hills, Massachusetts 02181

NEW AND CHANGED INFORMATION

Extensive functional descriptions and programming information for the Model 1250 have been added. New information and new peripheral devices for all Series 200 processors have been incorporated. Likewise, the information carried over from revision 1 has been extensively updated to correct technical errors and to enhance its clarity.

New information and changes added to this publication since the last edition are indicated below by page number and item.

Page	Item(s)	Page	Item(s)
1-8	Tables 1-1 and 1-2	8-62	Note 4
	Para. 3	8-63	Para. 3 and Table 8-15
1-9	Table 1-3 and	8-64	Note 4
	Para. 2	8-67	Note 2
1-11	Para. 2 and 3	8-82	Note 3
1-12	Table 1-7	8-85	Notes 6, 7, and 8
1-22	Note 5	8-86	All
1-23	Note 6 and Para. 4	8-87	All
1-24	All	8-88	All
2-4	All	8-89	All
2-5	All	8-90	All
2-6	Para. 1	8-92	Note 7
2-7	Table 2-2	8-94	All
2-10	Footnote 2	8-96	Note 3
2-17	Table 2-4	8-100	Note 4
5-19	Para. 5	8-111	Table 8-23
6-8	Para. 3 - Note	8-125	Note 1
8-15	Note 5	B-9	Table B-9
8-17	Note 5	C-9	Table C-2
8-21	Note 4	C-10	Table C-2
8-23	Note 4	C-11	Table C-3
8-24	Note 7	F-4	Table F-1
8-26	Note 9	F-5	Table F-1
8-43	Note 3	F-6	Table F-1
8-55	Example	G-2	Para. 5
8-58	Table 8-12	G-3	Para. 3, 4, and 5
8-60	Notes 4 and 5	Appendix H	All

TABLE OF CONTENTS

Page

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

Page

Section VIII (cont)

TABLE OF CONTENTS (cont)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (cont)

LIST OF TABLES

LIST OF TABLES (cont)

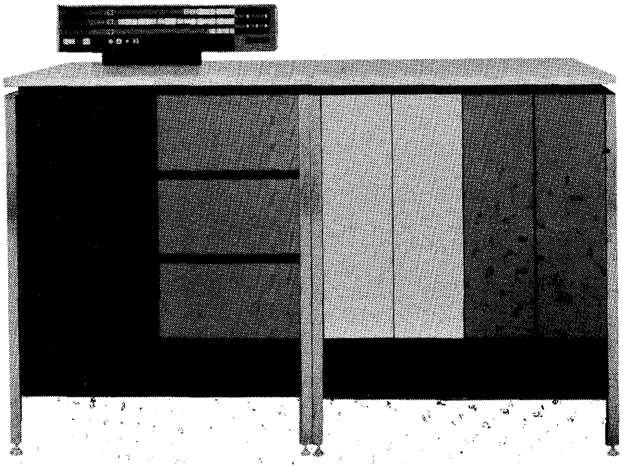
1

SERIES 200 COMPONENTS

Honeywell's Series 200 Data Processing System is a set of modularly designed, compatible models, five of which — the Models 200, 1200, 1250, 2200, and 4200 — are the subject of this manual. Each model consists of two basic elements: a central processor, and an array of peripheral devices connected to that processor. The peripheral equipment in the system can be attached to any processor, and the number of connectable devices is limited only by the number of unit power loads and peripheral address assignments available with the particular processor.

The initial member of Series 200 was the Model 200. The capabilities of the Model 200 processor have twice been extended since its introduction. Thus, seven central processors are described herein: the three processors of Model 200 (Types 201, 201-1, and 201-2); the Type 1201; the Type 1251; the Type 2201; and the Type 4201. The processing power of any one of these types can be increased at any time by the addition of peripheral devices and/or optional hardware features. This section describes: (1) the two basic elements of a Series 200 model (processor and peripheral devices); (2) the manner in which these elements communicate with one another; and (3) the expansion of processing power that is possible through the addition of optional hardware features to a processor.

CENTRAL PROCESSOR



The central processor is the computing and control center of a Series 200 model; instructions processed within the central processor control the operations of the entire computer. A Series 200 processor is functionally divided into three units: storage, control, and arithmetic. The storage unit provides magnetic core storage for both the program instructions and the data to be processed according to these instructions; it is also used to contain the resultant data. The control unit directs the operation of the entire computer by selecting, interpreting, and controlling the execution of all program instructions. It controls not only the flow of information within the central processor but also the flow of data between the central processor and all peripheral equipment. The arithmetic unit performs such operations as addition, subtraction, multiplication, division, and comparison, as directed by the control unit.

Included as a part of the central processor is a control panel (see Figure 1-1) which provides for easy communication between an operator and the computer. By using various control switches, the operator can start and stop the machine and can load and interrogate memory locations. The control panel also includes from four to eight SENSE switches which may be used in conjunction with programmed instructions to stop processing or to select predetermined program paths. The use of these switches increases the flexibility of a program, allowing it to be used in several different applications.

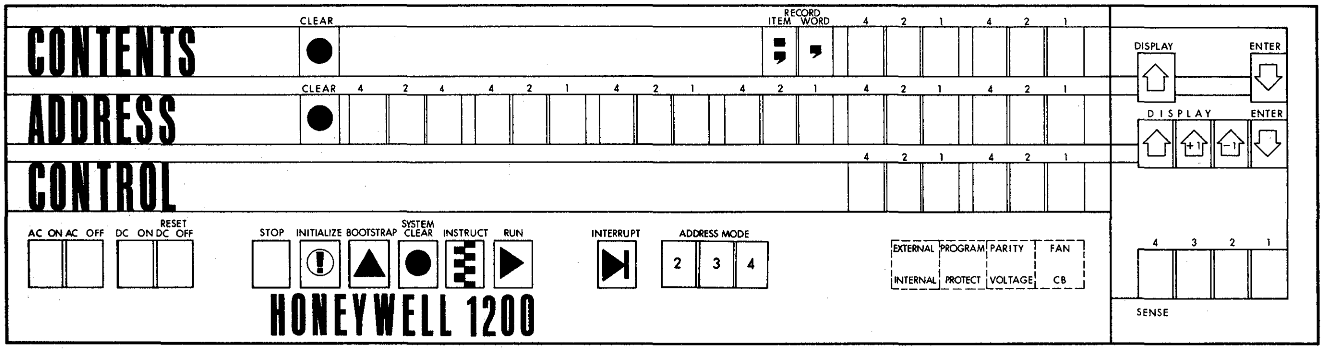


Figure 1-1. Type 1201 Control Panel

Another communication medium between the operator and the central processor is the Type 220 console, of which two versions are available. The Type 220-1 Console (Figure 1-2) contains a typewriter which may be used as a peripheral device, operating under program control, or as a logging typewriter by which the operatot can make essential notes about the program in progress. The central processor control panel remains situated on the processor cabinetry and is used for the functions described above.

In the Type 220-3 Console (Figure 1-3), most of the control panel functions, including that of direct access to the processor, are performed by means of the console typewriter. In addition, the typewriter can perform the peripheral and logging functions described for the Type 220-1. The central processor control panel is replaced by a smaller control panel containing only the main power switches, the SENSE switches, and certain check condition indicators which are located in the bottom row of the control panel shown in Figure 1-1. The Type 220-3 control panel contains additional indicators used with the Storage Protect Feature (see page 1-23) and the additional SENSE switches used with the larger Series 200 processors.

Figure 1-2. Type 220-1 Console

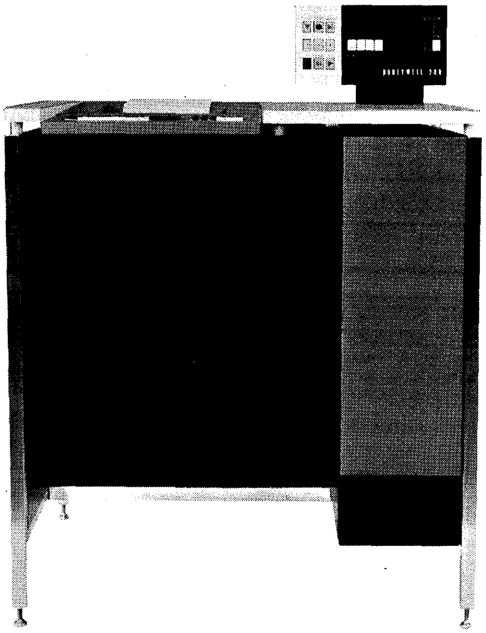


Figure 1-3. Type 220-3 Console

STANDARD PROCESSING MODE

The central processor performs arithmetic and logical operations as directed by the instructions of an internally stored program. These instructions are read into memory from an input medium such as punched cards, magnetic tape, or punched paper tape. Control circuitry within the processor then selects, interprets, and executes these instructions. Normally, the instructions are executed sequentially. Branch instructions are provided, however, which make it possible to skip over a group of instructions or otherwise change the sequence of the program.

INTERRUPT PROCESSING MODE

Sequential instruction execution is changed temporarily whenever the central processor is interrupted. Any one of four sources can "demand" access to the central processor by generating an interrupt signal, which turns on a central processor interrupt indicator. Once an interrupt indicator is detected as being on, a hardware response is made: information concerning the current status of the processor (including the setting of the sequence register) is stored, and a branch is made to a stored routine which identifies and services the demand. Thus, programmed tests need not be made to detect the presence of an interrupt condition — the entire process of detecting and responding to an interrupt signal is an automatic hardware function. After the stored service routine has been executed, control is returned to the interrupted routine at the point where the interruption occurred and the previous status is restored. Two kinds of interrupts can occur in the system: external interrupts and an internal interrupt. A detailed description of interrupt functions and programming for interrupt processing is presented in Appendix D.

External Interrupts

The three sources of external interrupts are:

1. Peripheral Control — The control connected to any Series 200 peripheral device can generate an interrupt signal under program control (peripheral controls are described on page 1-7; peripheral control interruption is described in Appendix D. For instance, a data communication control which services one or a number of communication lines and devices may generate a real-time demand on central processor time to handle a customer inquiry from a remote terminal. The current operations of the processor are temporarily interrupted so that the inquiry may be serviced. A routine to read the inquiry and to answer the question from a stored customer file is automatically executed, and a response is sent back to the terminal.

2. Operator's Control Panel or Console — The operator can interrupt the central processor by pressing the INTERRUPT button on the control panel or console.[1] The source of such "on-site" interruptions is made available to the program by the execution of a single instruction at the beginning of the interrupt service routine.

3. Program Instruction — One instruction in the Series 200 repertoire, the Monitor Call instruction, is used to generate an interrupt condition.[1] For programming convenience, the activation (or "calling") of the monitor program can be accomplished by means of this instruction.

Internal Interrupt

If a central processor contains the Storage Protect Feature (Types 1201, 1251, 2201, and 4201 only), an internal interrupt condition, caused by certain violations of a protected memory area or attempts to address nonexistent memory locations, can also occur. Internal interruptions are of lower priority than external interrupts, so that a processor executing an external interrupt service routine does not respond to an internal interruption until the routine is completed. Processing of internal interrupts is described in Appendix E.

ADDRESSING MODES

Due to the unique binary addressing system used in referencing the individual core storage locations within the central processor, an address portion of a machine-language instruction can occupy two, three, or four characters of memory. The number of character positions employed is controlled by two instructions: the assembly control instruction ADMODE, and the Change Addressing Mode (CAM) instruction. Any core storage address can be referenced in any addressing mode by having the central processor prefix the address expressed in the instruction with a binary value previously set in an address register. Thus, the programmer has the ability to set the address registers to some high module, switch to the two-character addressing mode, and still continue to address that module. This utilization of the smallest number of character

[1] The Type 201 and 201-1 processors cannot be interrupted by sources 2. and 3. above.

positions to express any core storage address results in a reduction in the amount of memory required for a particular program.

ITEM-MARK TRAPPING MODE

The item-mark trapping mode, which can be set via the CAM instruction, causes the processor to treat and execute any instruction containing an item-marked op code as if it were a Change Sequencing Mode (CSM) instruction, which results in a transfer of control to an instruction stored at a prespecified location. This processing mode is used extensively in Liberator systems and can also be used to control program branching.

PROCESSING POWER

The power of any processor within Series 200 can be defined as the sum of its main memory size, its internal speed, its degree of peripheral simultaneity, and the number of optional features which may be added to it.

Main memory size within the Models 200/1200/1250/2200/4200 ranges from a minimum of 2, 048 character locations (Types 201 and 201-1) to 524, 288 locations (Type 4201). Figure 1-4 shows the modular main memory sizes of the seven processor types.

The internal speed of a processor is measured in terms of a memory cycle (i. e., the time required to read and restore the contents of a unit location). The unit location used by processors other than the Type 4201 is a single, six-bit character location. The unit location of the Type 4201 is four successive character locations that contain a four-character word. Memory cycle speeds range from two microseconds per character to 750 nanoseconds per four-character word (see Figure 1-5).

Peripheral simultaneity is a key feature of Series 200 processors. Among the processors described in this manual, from 3 (Model 200 processors) to 16 (Type 4201 processor) simultaneous input/output operations can be performed concurrently with internal computing (see Figure 1-6).

A number of optional features can be included in the Series 200 processors to provide complete flexibility in specializing any one processor to a user's particular application. Since some of these features refer to the peripheral capabilities of a processor, they are summarized at the conclusion of this section.

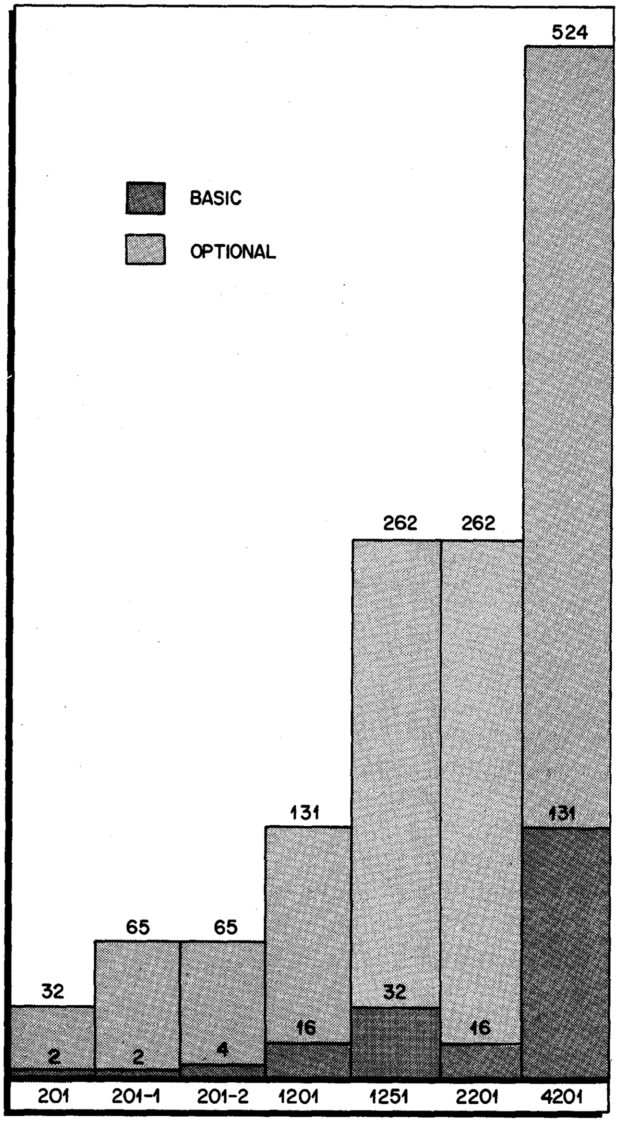
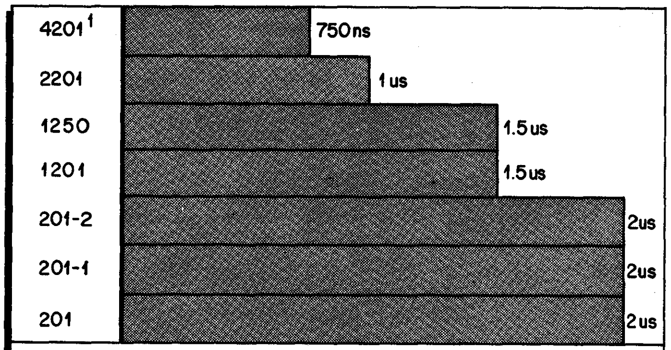


Figure 1-4. Main Memory Size



[1] The Type 4201 moves four successive characters in 750 nanoseconds. Anyone of these characters is thereby moved in 188 nanoseconds.

Figure 1-5. Main Memory Speed

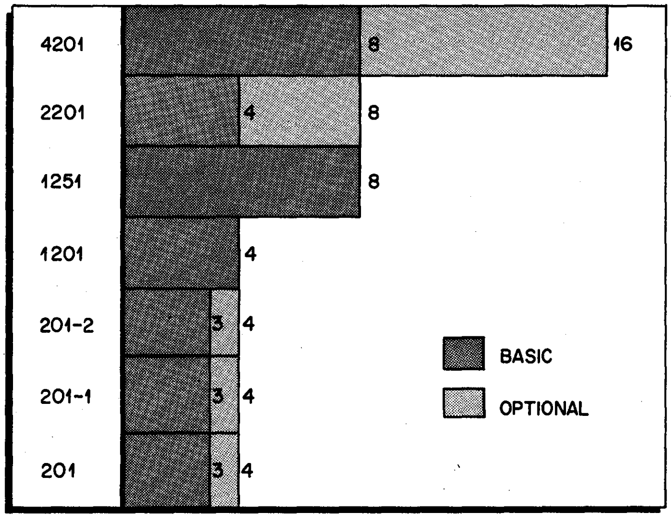


Figure 1-6. Peripheral Simultaneity (Number of Read/Write Channels Available to Processors)

PERIPHERAL EQUIPMENT

The array of peripheral devices available with Series 200 processors includes over 40 units: console typewriters, punched card equipment, high-speed printers, magnetic tape units, paper tape equipment, random access drum units, disk devices, MICR reader-sorters, multiple tape listers, teller terminals, visual information projection units, and various data communication controls and remote terminals. Also included are computer-to-computer adapters, an interval timer, a time-of-day clock, and controls for optical source-document readers, optical journal readers, digital plotters, and a bill feed printer.

Information is transferred between any one of these devices and the central processor by means of a single stored-program instruction — the Peripheral Data Transfer instruction described in Section VIII. By coding various control characters in this instruction, the programmer specifies the direction of data transfer (into or out of the processor), the specific device involved in the transfer, the data path over which information is to be transferred, and any other information necessary to define the input/output operation (e.g., the number of lines to be spaced during printer operations). The actual communication with the central processor is not made by the particular peripheral device but by the peripheral control connected to that device.

PERIPHERAL CONTROL

A peripheral control regulates the transfer of data between a processor and a peripheral device. The control compensates for the difference in the data transfer rates of the processor and the peripheral device by temporarily storing each character of transmitted information until either the processor or the device is ready to receive the character. The control also converts each character into the code used by the intended recipient (e.g., the card reader control converts a character from Hollerith code to the internal six-bit code of the central processor). As each character is transferred to the control, it is also checked for accuracy by the control. One particularly significant feature of the peripheral control is that it operates independently of the central processor and requires access to the main memory only when information transfers are performed. In particular, all of the previously mentioned activities of the control — temporarily storing, converting, and checking the information — do not involve the central processor in any way. When each character of information is transferred, one main memory cycle is allocated for the transfer.

Some peripheral devices require one peripheral control per device (e.g., a card reader). Other devices can be connected in multiple fashion to a single peripheral control (e.g., up to eight 1/2-inch magnetic tape units can be directed by a single control). The number of Series 200 devices connectable to a peripheral control is shown in Tables 1-1 through 1-10 on the following pages. The information listed under "Unit Loads" and "Address Assignments" in these tables is used in determining the number of peripheral controls that can be connected to a Series 200 processor, as explained on page 1-17.

PUNCHED CARD EQUIPMENT

Series 200 includes a wide variety of peripheral devices not only of different kinds, but also on several performance levels for the same kind. For instance, four different punched card units are offered: two card readers, a card punch, and one reader/punch. Table 1-1 lists the card devices available within Series 200. Note that a card device requires either one or two "unit loads," depending on the number of functions the device performs.

Table 1-1. Punched Card Equipment

Device		Read/Punch Speed	No. Devices Per Control	No. Unit Loads Required by Control & Device	Address Assignments
Type	Function				
223	Card Reader	800 cards/minute	1	1	1
223-2	Card Reader	1050 cards/minute	1	1	1
214-1	Card Punch	100-400 cards/minute	1	1	1
214-2	Card Reader/Punch	Read: 400 cards/minute Punch: 100-400 cards/minute	1	1	2

HIGH-SPEED PRINTERS

Six types of printers (see Table 1-2) produce printed reports, listings, etc., at speeds which vary from 450 to 1,300 lines per minute. Processed information is printed from any programmer-assigned area in memory. A single program instruction — the Move Characters and Edit instruction — allows the programmer to punctuate the output date, suppress zeros, and insert identifying symbols in the data prior to printing.

Print Buffer

With the addition of the Print Buffer (Feature 036), the amount of central processor memory cycles required for data transfer to the printer is reduced to less than 1%. Thus, more than 99% of the central processor time is available for program execution. This feature is available only for the Type 222-3, -4, -5, and -6 Printers.

MAGNETIC TAPE UNITS

Magnetic tape is a compact and highly versatile medium for the storage of programs and data files. Two complete families of industry-acclaimed tape units are available with Series 200 processors (see Table 1-3): 1/2-inch tape units (10 types) transfer data at speeds ranging from 4,800 to 96,000 characters per second; three types of 3/4-inch tape units read/write from 32,000 to 88,800 characters per second. The capability of processing nine-track, 1/2-inch tape is also provided.

1200 BPI Recording Density

The 1200-bits-per-inch recording density (Feature 054) provides the Type 204B-9 Magnetic Tape Unit with the capability of reading and writing data at a density of 1200 bits per inch (bpi) on Dupont Crolyn magnetic tape. The 1200-bpi recording density enables the 204B-9 to achieve a transfer rate of 144,000 characters per second.

Table 1-2. High-Speed Printers

Type	Print Speed	No. Printers Per Control	No. Unit Loads Required by Control & Device	Address Assignments
222-1 (96 print positions)	650-1, 300 lines/minute	1	1	1
222-2 (108 print positions)	650-1, 300 lines/minute	1	1	1
222-3 (120 or 132 print positions)	650-1, 300 lines/minute	1	1	1
222-4 (120 or 132 print positions)	950-1, 266 lines/minute	1	1	1
222-5 (120 or 132 print positions)	450 lines/minute	1	1	1
222-6 (120 or 132 print positions)	1100 lines/minute	1	1	1
229[1] (120 or 132 print positions)	400 lines/minute	1	1	1

[1] Restricted to educational institutions.

Table 1-3. Magnetic Tape Units

Type	Data Transfer Rate	No. Devices Per Control	No. Unit Loads Required by Control & Devices	Address Assignments
1/2-Inch Magnetic Tape Units				
204B-1 204B-2	7,200/20,000 characters/second	1-8	2	2
204B-3 204B-4	16,000/44,500 characters/second	1-8	2	2
204B-5	24,000/66,700 characters/second	1-8	2	2
204B-7	20,000/28,800 (or 7,200/28,800) characters/second	1-8	2	2
204B-8	44,500/64,000 (or 16,000/64,000) characters/second	1-8	2	2
204B-9	66,700/96,000 (or 24,000/96,000 or 66,700/144,000 or 96,000/144,000) characters/second	1-8	2	2
204B-11 204B-12	4,800/13,300 characters/second	1-4	2	2
204C-13 204C-14	28,800 characters/second	1-2	2	2

Table 1-3 (cont). Magnetic Tape Units

Type	Data Transfer Rate	No. Devices Per Control	No. Unit Loads Required by Control & Devices	Address Assignments
3/4-Inch Magnetic Tape Units				
204A-1	32,000 characters/second	1-4	2	2
204A-2	64,000 characters/second	1-4	2	2
204A-3	88,800 characters/second	1-4	2	2

DISK PACK DRIVES

Honeywell disk pack drives combine the desirable features of magnetic tape and magnetic disk storage — unlimited shelf storage and fast random access. This is made possible by the use of removable disk packs which may be recorded on, stored indefinitely (like magnetic tape), and rapidly reinserted in an on-line drive. The various disk pack drives are listed in Table 1-4.

Table 1-4. Disk Pack Drives

Type	Data Storage Capacity Per Drive	Data Transfer Rate	No. Devices Per Control	No. Unit Loads Required by Control & Devices	Address Assignments
258	4.6 million characters	208,333 characters/second	1-8	1	2
259	9.2 million characters	208,333 characters/second	1-8	1	2
259A[1]	9.2 million characters	147,500 characters/second	1-8	1	2
259B	9.2 million characters	147,500 characters/second	1-8	1	2

[1] Used in systems with Type 201 and 201-1 Central Processors.

DISK FILES

The Honeywell disk files are fixed-disk storage devices which provide an extremely high on-line storage capacity (see Table 1-5). A single disk file subsystem's capacity may amount to over 1.2 billion characters. Any on-line data track can be located in a maximum time of 120 milliseconds, and data can be transferred at a rate of 190,000 characters per second.

Table 1-5. Disk Files

Type	Data Storage Capacity Per File	Data Transfer Rate	No. Devices Per Control	No. Unit Loads Required by Control & Devices	Address Assignments
261	150 million characters	190,000 characters/second	1-8	1	2
262	300 million characters	190,000 characters/second	1-4	1	2

RANDOM ACCESS DRUMS

The Series 200 drum storage capability features a drum control which can direct from one to eight magnetic drums, each capable of storing 2.6 million characters of information (see Table 1-6). Thus, a single drum subsystem can have a total capacity of over 20 million characters. Any record stored on the drum can be located in 27 milliseconds (average) and can be transferred at the rate of 111,000 characters per second.

Table 1-6. Random Access Drum Units

Type	Data Storage Capacity Per Drum	Data Transfer Rate	No. Devices Per Control	No. Unit Loads Required by Control & Devices	Address Assignments
270A-1 through 270A-8	2.6 million characters	111,000 characters/second	1-8	1	2

HIGH-SPEED DRUMS

The high-speed drums are fixed-head storage devices which offer high speed performance with fast access time. Up to four devices can be operated with a single drum control, and thus a control's capacity may amount to over 16.8 million characters. Any record stored on the drums can be located in 8.6 milliseconds (average).

Angular Position Indicator

Features 072 and 073 (Angular Position Indicator) provide for optimum addressing of the Type 265/266 and 267 High-Speed Drums, respectively. Information is provided at any given time as to the current drum position relative to 360 degrees of rotation. Under heavy load conditions with many demands waiting to be executed, the average access time of the drums may be substantially reduced.

Table 1-7. High-Speed Drums

Type	Data Storage Capacity Per Drum	Data Transfer Rate	No. Devices Per Control	No. Unit Loads Required by Control & Devices	Address Assignments
265[1]	2.1 million characters	300,000 char./second	1-4	1	2
266[1]	4.2 million characters	300,000 char./second	1-4	1	2
267[2]	4.2 million characters	1,200,000 char./second	1-4	1	2

[1] Used only in systems with Type 1251, 2201, or 4201 Central Processors.

[2] Used only in systems with Type 4201 Central Processors equipped with the High-Speed Third Sector (Feature 4215).

PAPER TAPE EQUIPMENT

Paper tape is an ideal medium for recording data which originates at locations distant from a central Series 200 installation and, as such, becomes particularly significant in data communication networks. A variety of standard commercial codes may be used with this relatively inexpensive medium. Two paper tape devices are offered in Series 200 (see Table 1-8).

Table 1-8. Paper Tape Equipment

Device		Data Transfer Rate	No. Devices Per Control	No. Unit Loads Required by Control & Devices[1]	Address Assignments
Type	Function				
209-2	Paper Tape Reader	600 characters/second	1	2	1
210	Paper Tape Punch	120 characters/second	1	2	1

[1] The total power requirement for the combination of a 209-2 reader and a 210 punch is 3 unit loads

DATA COMMUNICATION EQUIPMENT

The immediate and automatic response to an external interrupt by the Series 200 processor is described on page 1-3. A common source of external interruption is a communication control. These controls allow the Series 200 processor to communicate with distant locations (e.g., branch offices, warehouses, etc.) by receiving and transmitting data over toll and leased lines. Four kinds of communication controls are available in Series 200: (1) two types of single-channel controls transfer entire messages over single lines; (2) three types of multi-channel controls transfer messages character-by-character over as many as 63 different lines; (3) two types of message-mode multi-channel controls transfer entire messages over a maximum of 63 lines; and (4) two types of controls serve as interfaces with the Air Force AUTODIN network. All controls are adaptable to a broad selection of lines, speeds, and terminal devices. One such terminal device is Honeywell's Data Station (see Table 1-9).

Table 1-9. Data Communication Equipment

Device					
Type	Function	Data Transfer Rate	No. Devices Per Control	No. Unit Loads Required by Control & Devices	Address Assignments
		Communication Controls			
281-1 and -2	Single-Channel Controls	Up to 5,100 characters/ second	1 line	1[1]	2
286-1, -2, and -3	Multi-Channel Controls	Up to 300 characters/ second/line	1-63 lines	2	2
286-4, -5[2]	Message-Mode, Multi-Channel Controls	Up to 7,000 characters/ second (all lines)	1-63 lines	2	2
287	AUTODIN Communication Control	Up to at least 4,800 baud	1 line	2	2
287-1	(USASCII) AUTODIN Communication Control	Up to at least 4,800 baud	1 line	2	2
		Remote Terminal Device			
288-1	Data Station Central Control	120 characters/second	n/a	n/a	n/a
288-3[3]	Data Station Central Control	300 characters/second	n/a	n/a	n/a
289-2	Data Station Page Printer & Keyboard	10 characters/second	n/a	n/a	n/a
289-2A	Keyboard	10 characters/second	n/a	n/a	n/a
289-3	Data Station Page Printer & Keyboard	40 characters/second	n/a	n/a	n/a
289-4	Data Station Paper Tape Reader	120 characters/second	n/a	n/a	n/a
289-5	Data Station Paper Tape Punch	120 characters/second	n/a	n/a	n/a
289-7	Data Station Card Reader	143 characters/second	n/a	n/a	n/a
289-8	Data Station Optical Bar Code Reader	50 characters/second	n/a	n/a	n/a
289-9[3]	Remote Line Printer	300 characters/second	n/a	n/a	n/a

[1] The Type 281-2 control requires two unit loads.

[2] Not available for use on the Type 201 and 201-1 Central Processors.

[3] Both required for operation of either unit.

A major requirement of many communication networks (e.g., inquiry handling or message switching applications) is fast access to a stored file. Files may sometimes be stored in main memory, but for large files main memory storage is economically unfeasible. File storage units (i.e., the disk pack drives or drum units) fulfill the requirements of these applications.

A typical data communication network is shown in Figure 1-7. The pertinent components of this system are: (1) a Type 201-2 processor; (2) a Type 259 Disk Pack Drive; (3) a Type 281 Communication Control; (4) two data sets[1]; and (5) a Honeywell Data Station, the remote terminal device. Two particular devices connected to the Data Station are used in this example: a keyboard by which the inquiry is transmitted to the central processor, and a page printer which prints the answer to the inquiry in readable form.

CONSOLE EQUIPMENT

Characteristics of the Type 220 consoles, described previously on page 1-2, are listed in Table 1-10.

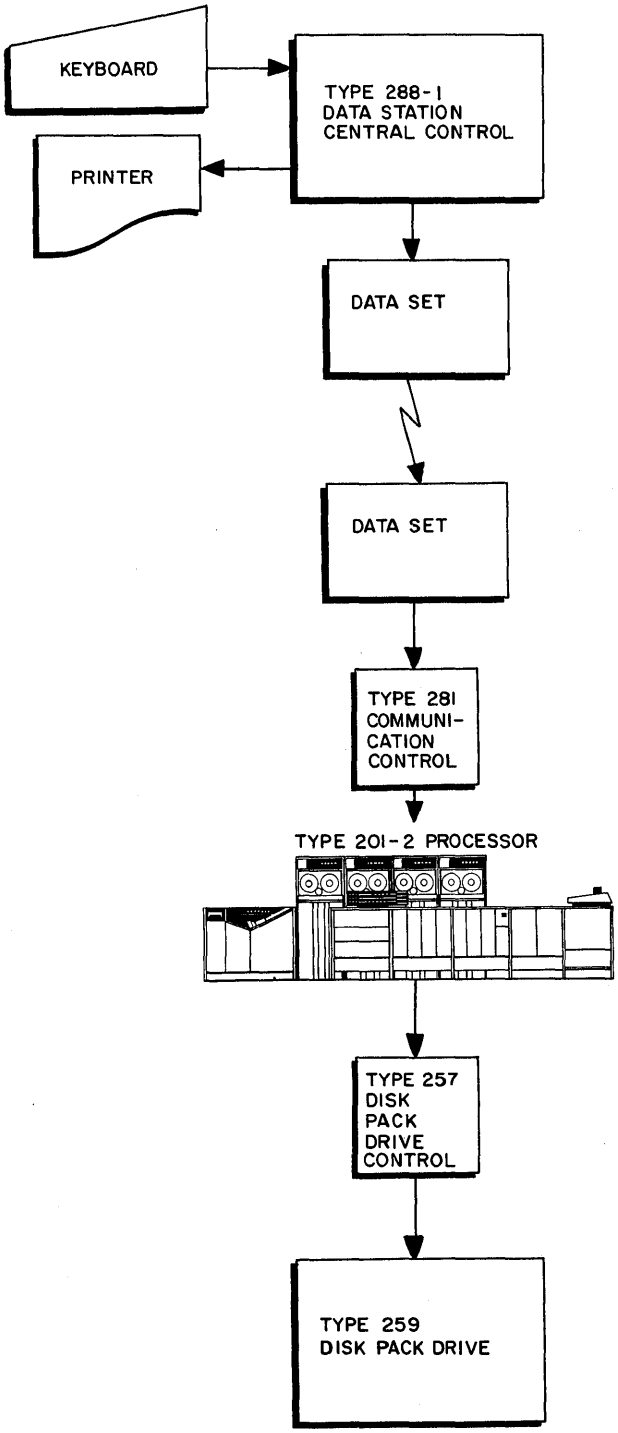
Table 1-10. Console Equipment

Device					
Type	Function	Data Transfer Rate	No. Devices Per Control	No. Unit Loads Required by Control & Devices	Address Assignments
220-1, -3	Operator's Console	Typing Speed (input); or 10 char./sec. (output)	1	1	2

VISUAL INFORMATION PROJECTION DEVICES

Cathode-ray tube (CRT) display units — for businesses requiring instantaneous visual access to data stored in computer files — are available to the Series 200 user. These devices operate on line to the computer, either locally via direct physical connection or from remote locations via communication facilities. As shown in Table 1-11, the display devices provide a variety of keyboard arrangements: numeric, numeric/block alpha, and typewriter. An 8-bit code (7-bit USASCII plus parity) is used for synchronous transmission and a 10-bit code (7-bit USASCII plus parity and start and stop bits) for asynchronous transmission.

[1] A data set is required to convert the data signals used by the communication control to signals acceptable for transmission over communication lines.


KEYBOARD
PRINTER
TYPE 288-1 DATA STATION CENTRAL CONTROL
DATA SET
DATA SET
TYPE 281 COMMUNICATION CONTROL
TYPE 201-2 PROCESSOR
TYPE 257 DISK PACK DRIVE CONTROL
TYPE 259 DISK PACK DRIVE

1. Customer inquiry is typed on keyboard in form of a coded message.

2. Message signals are converted to a form acceptable for transmission line.

3. Message is transmitted over transmission line.

4. Message signals are reconverted.

5. Control generates interrupt signal and transfers incoming message to preassigned memory location as directed by interrupt service routine.

6. Stored interrupt service routine interprets message and issues instructions to read and update the customer's record in a file stored in Type 259 Disk Pack Drive.

7. Type 257 control directs the execution of the instructions issued by the stored interrupt program.

8. Customer's record is read and updated according to instructions. Record is read into preassigned location in interrupt routine (in central processor memory), from which the answer to the inquiry is sent back to the Data Station. (Answer to inquiry is printed by page printer.)

Figure 1-7. Customer Inquiry Handling via Typical Communications Network

Table 1-11. Visual Information Projection Devices

Device		Data Transfer Rate	No. Devices Per Control	No. Unit Loads Required by Control & Devices	Address Assignments
Type	Function				
303	Display Station (Typewriter keyboard)	1200-2400 baud	12 for basic control, plus 12 for each expansion module	n/a	n/a
311	Display Station (15-key block numeric keyboard)				
312	Display Station (43-key numeric/block alpha keyboard)				
304	Display Station (Navcor electronic typewriter keyboard)				
317	Display Station (no keyboard)				
318	Display Station (no keyboard)				

TELLER TERMINAL EQUIPMENT

Honeywell Teller Terminal equipment permits more efficient banking procedures through on-the-counter, on-line processing of all teller-assigned transactions. The Type 370 Teller Terminal is used by the teller for all his bank transactions, and a remote transceiver transmits transaction information between the Type 370 and the computer. Data is transmitted asynchronously via a modified USASCII-type code permitting combinations of similarly coded peripheral devices to share common networks. This code consists of a start bit, seven data bits, an odd parity bit, and a stop bit. Specifications of the Type 370 are shown in Table 1-12.

Table 1-12. Teller Terminal Equipment

Device		Data Transfer Rate	No. Devices Per Transceiver	No. Unit Loads Required by Transceiver & Devices	Address Assignments
Type	Function				
370	Teller Terminal	120 characters/second	1, 2, 6, or 10	not applicable	not applicable

ADDITIONAL PERIPHERAL DEVICES

A number of other peripheral devices are included in the Series 200 line. General characteristics of these devices are shown in Table 1-13.

Table 1-13. Additional Peripheral Devices

Device		Performance	No. Devices Per Control	No. Unit Loads Required	Address Assignments
Type	Function				
212	On-Line Adapter	120,000 characters/second	1	1	1
212-1	Central Processor Adapter	167,000 characters/second	1	2	2
213-3	Interval Timer	Range: 100 microseconds to 200 milliseconds	1	1	1
213-4	Time-of-Day Clock	Range: 00:00:00.0 to 23:59:59.9 (hours, minutes, seconds, and tenths of seconds)	1	1	1
232	MICR Reader-Sorter and Control	Up to 600 documents/minute	1	1	1
233-2	MICR Reader-Sorter Control for Burroughs B103	Up to 1,560 documents/minute	1	1	1
234	Plotter Control for Calcomp Plotters	Plotting Speed: Up to 300 increments per second (in any of eight directions)	1	1	1
235	Optical Journal Reader Control	26 or 52 lines/second	1	1	1
237	Bill Feed Printer Control	600 lines/minute; or up to 800 cards/minute	1	2	2

PERIPHERAL DATA TRANSFER OPERATION

One of the major features of Series 200 is the degree of peripheral simultaneity that can be achieved by the various processors. The Model 200 processors (Types 201, 201-1, and 201-2) and the Type 1201 processor can perform up to four peripheral operations simultaneously; the Type 1251, 2201, and 4201 processors may perform as many as six, eight, and sixteen simultaneous peripheral operations, respectively. While all these operations are being executed, the central processor continues its internal processing. The ability to perform simultaneous peripheral operations derives from an internal unit of the central processor — the input/output traffic control — which guarantees a peripheral control access to main memory when data is to be transferred. The manner in which the traffic control does this is explained in Section II. The data path used by the traffic control to transfer data (see Figure 1-8) is described below.

Peripheral Addresses and Unit Loads

When installed in a Series 200 computer system, peripheral controls (and their associated devices) are permanently connected to the system. Each control is assigned one or two addresses, depending on the number of directions in which it can transfer data. It is by these peripheral addresses that the controls are designated in input/output instructions. For example, a card reader and its associated control can transfer data in only one direction — into the central processor. The reader control is therefore assigned one address by which it is always designated in an instruction. A combination card reader/card punch and control can transfer data in two directions — into and out of the processor. It is thereby assigned two addresses: one address is used to specify an input (card read) operation, while the other is used to specify an output (card punch) operation.

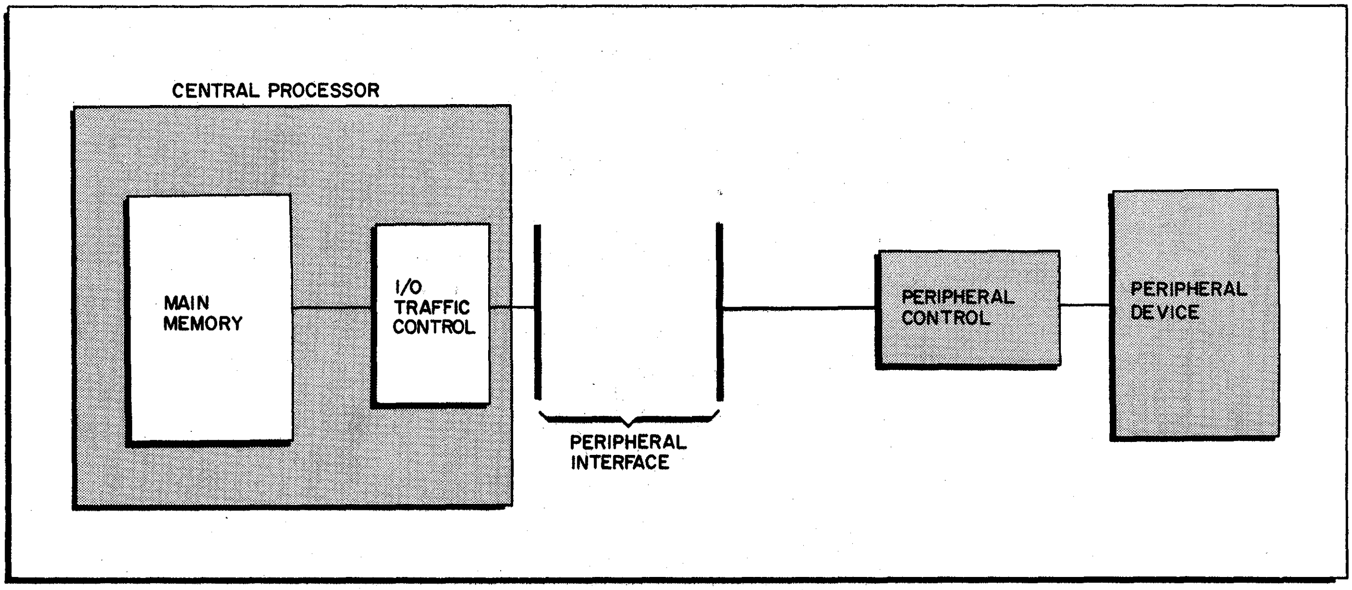


Figure 1-8. Basic Input/Output Data Path

The number of peripheral controls which a Series 200 processor can accommodate depends upon four factors: (1) the number of "unit loads" of power required by the controls to be connected; (2) the number of unit loads of power available from the processor; (3) the number of peripheral addresses required to operate the controls; and (4) the number of address assignments which the processor provides. A peripheral control may require either one or two unit loads of power and either one or two addresses. The numbers of unit loads and address assignments available with each Series 200 processor are shown in Figure 1-9. The numbers of unit loads and address assignments required by each peripheral control are shown in the preceding summary tables of the peripheral equipment (Tables 1-1 through 1-13).

Read/Write Channel

Note that the permanent connection established in Figure 1-8 is incomplete: there is no connection across the peripheral interface. The input/output data path is completed by one or more "read/write channels," inserted in the data path when the input/output instruction is executed. (More than one read/write channel is sometimes necessary in order to accommodate the high data transfer rates of some devices.) A read/write channel is not permanently connected to any peripheral control but is assigned by the programmer to specify the data path between a control and the processor.

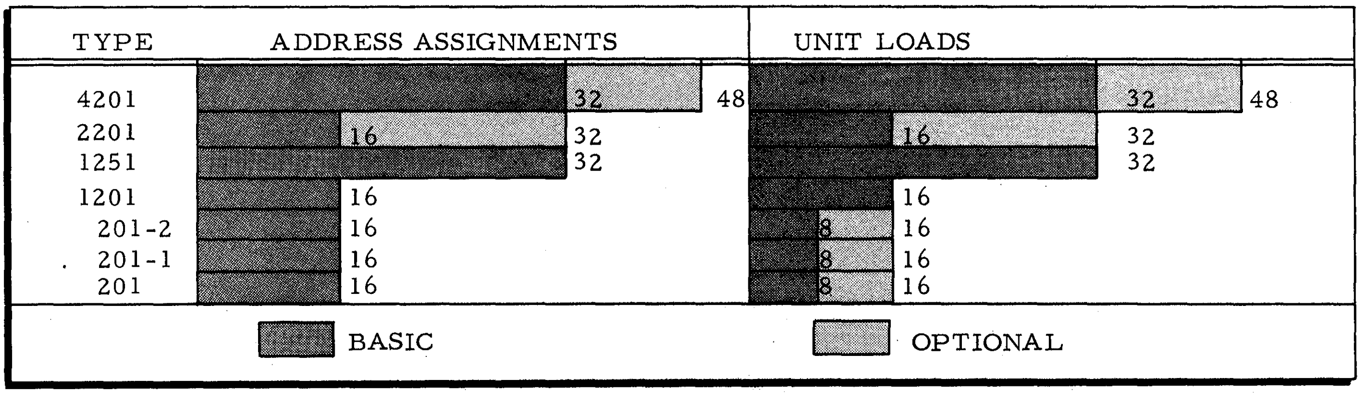


Figure 1-9. Address Assignments and Unit Loads Available in Series 200 Processors

When the programmer codes an input/output instruction, he specifies among other things the address of the peripheral control that is to send or receive data and the read/write channel(s) over which the data transfer is to take place. When the instruction is executed, the specified read/write channel is automatically inserted in the peripheral interface. For example, Figure 1-10 shows the data path formed during the execution of an input/output instruction in which the programmer specifies that the card reader control is to transfer data over read/write channel 2 (RWC 2). The specified channel remains in the interface only for the duration of the card read operation. When the data transfer terminates, RWC2 is automatically removed from the interface and is available for reassignment by another instruction.

Read/write channels are the key to the achievable simultaneity in a Series 200 model: the number of read/write channels associated with a particular processor determines the number of peripheral operations that can be performed simultaneously by that processor (see Figure 1-6).

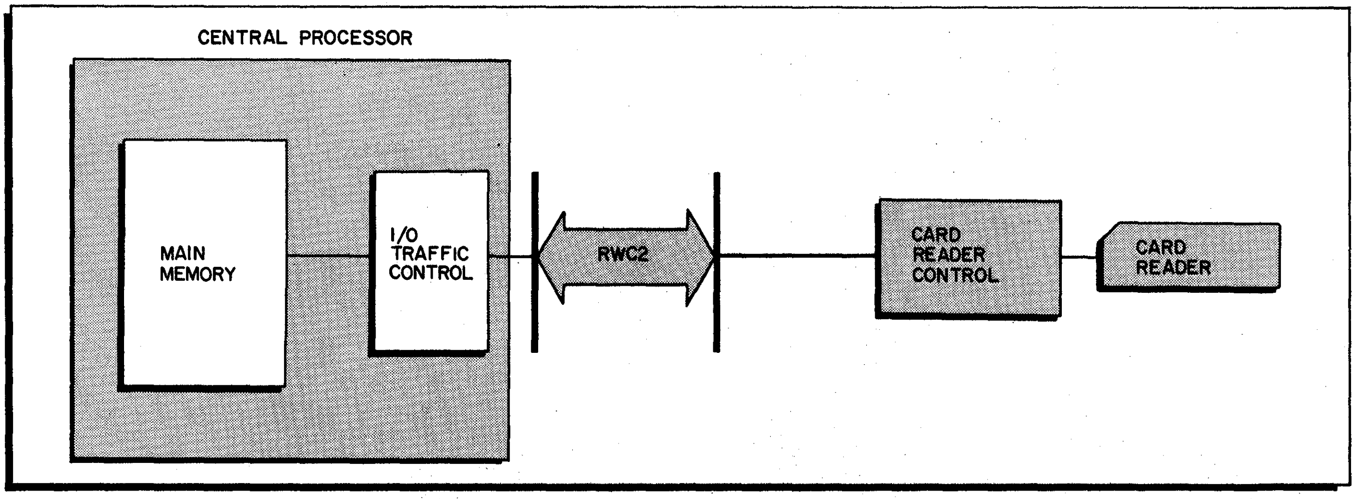


Figure 1-10. Data Path During Card Read Operation

OPTIONAL FEATURES

Table 1-14 lists the various features that can be added to the Series 200 processors described in this manual. This table illustrates the realistic design principle of Series 200: a Series 200 model can be specialized to meet the individual user's application; the application is not compromised to meet the design of the model.

Certain features optional with some processors are standard with other larger types. This is also part of the realistic approach to system development. Particularly significant is the fact that specialization of a Series 200 model can occur at any time (not just at installation time) to meet any increased workload or applications shift that might occur.

A summary description of the optional features is given below.

Table 1-14. Series 200 Optional Features

FEATURE		201	201-1	201-2	1201	1251	2201	4201
010	ADVANCED PROGRAMMING	n/a	n/a	OPT	▨	▨	▨	▨
011	ADVANCED PROGRAMMING	OPT	OPT	n/a	n/a	n/a	n/a	n/a
012	PROGRAM INTERRUPT	OPT	▨	▨	▨	▨	▨	▨
013	EDIT INSTRUCTION	OPT	OPT	OPT	▨	▨	▨	▨
015	8 ADDITIONAL UNIT LOADS	OPT	OPT	OPT	▨	▨	▨	▨
016	AUXILIARY READ/WRITE CHANNEL	OPT	OPT	OPT	▨	▨	▨	▨
0191	OPTIONAL INSTRUCTION PACKAGE	n/a	n/a	n/a	OPT	OPT	OPT	▨
1100A	SCIENTIFIC UNIT	n/a	n/a	n/a	OPT	OPT	OPT	n/a
1101	SCIENTIFIC UNIT	n/a	n/a	n/a	n/a	n/a	n/a	OPT
1114	STORAGE PROTECT	n/a	n/a	n/a	OPT	OPT	n/a	n/a
1115	16 ADDITIONAL UNIT LOADS & ADDRESS ASSIGNMENTS & 4 ADDITIONAL RWC'S	n/a	n/a	n/a	n/a	n/a	OPT	n/a
1116	16 ADDITIONAL UNIT LOADS & ADDRESS ASSIGNMENTS & 8 ADDITIONAL RWC'S	n/a	n/a	n/a	n/a	n/a	n/a	OPT
1117	STORAGE PROTECT	n/a	n/a	n/a	n/a	n/a	OPT	n/a
1118	EXTENDED MULTIPROGRAMMING & 8-BIT TRANSFER	n/a	n/a	n/a	n/a	n/a	n/a	OPT
1120	EXTENDED MULTIPROGRAMMING & 8-BIT TRANSFER	n/a	n/a	n/a	OPT	OPT	n/a	n/a
1121	EXTENDED MULTIPROGRAMMING & 8-BIT TRANSFER	n/a	n/a	n/a	n/a	n/a	OPT	n/a
4214A	TWO BUFFERED I/O SECTORS	n/a	n/a	n/a	n/a	n/a	n/a	OPT
4214B	TWO ADDITIONAL BUFFERED I/O SECTORS	n/a	n/a	n/a	n/a	n/a	n/a	OPT
4215	HIGH-SPEED THIRD SECTOR	n/a	n/a	n/a	n/a	n/a	n/a	OPT
	▨ STANDARD OPT OPTIONAL n/a NOT AVAILABLE							

ADVANCED PROGRAMMING

Two Advanced Programming Features increase the basic instruction repertoire of the Model 200 processors. Feature 011 is available with the Type 201 and 201-1 processors, and Feature 010 can be added to the Type 201-2 processor. Each feature includes the following capabilities (see Table 1-15):

1. Additional program instructions.
2. The ability to modify instruction addresses via indexed or indirect addressing (described in Section IV).
3. A "read reverse" capability with magnetic and paper tape units.

Table 1-15. Model 200 Advanced Programming Feature

FEATURE 010 (Type 201-2)	FEATURE 011 (Types 201 and 201-1)
Program Instructions	Program Instructions
1. Zero and Add 2. Zero and Subtract 3. Branch if Character Equal 4. Change Sequencing Mode 5. Change Addressing Mode (expanded version)[1] 6. Extended Move 7. Move and Translate 8. Branch on Character Condition (expanded version) 9. Branch on Bit Equal	1. Zero and Add 2. Zero and Subtract 3. Branch if Character Equal 4. Change Sequencing Mode 5. Change Addressing Mode[1] 6. Extended Move 7. Move and Translate 8. Branch on Character Condition (expanded version) 9. Load Control Registers[2]
Address Modification	Address Modification
1. Indexed addressing via 6 or 15 index registers[3] 2. Indirect addressing	1. Indexed addressing via 6 or 15 index registers[3] 2. Indirect addressing
Read Reverse	Read Reverse
Any Model 200 processor can read paper tape and 1/2-inch magnetic tapes in a reverse direction and transfer the information to the main memory in such a manner that it is oriented in the normal (forward) direction.	

[1] The Change Addressing Mode instruction is available in Type 201 or 201-1 processors which include either the Advanced Programming Feature or a main memory capacity greater than 4,096 characters. In the Type 201-2 processor, the use of this instruction with 2- and 3-character addressing is standard; however, its use with 4-character addressing and/or item-mark trapping requires the presence of the Advanced Programming Instructions.

[2] The Load Control Registers instruction, optional with the Type 201 and 201-1 processors, is included in the standard instruction repertoire of the Type 201-2 processor.

[3] The Type 201-1 and 201-2 processors with the Advanced Programming Feature contain 6 index registers in the three-character addressing mode and 15 index registers in the four-character mode. The Type 201 processor with the Advanced Programming Feature contains six index registers, regardless of addressing mode.

PROGRAM INTERRUPT

This feature, whose basic functions are described on page 1-3, is an optional feature for the Type 201 processor and is standard for all other processors described herein. A detailed description of program interruption, including conditions which must be present for an interrupt to occur, processor activities which are automatically performed when the interrupt takes place, and the programming of interrupt service routines, is given in Appendix D.

EDIT INSTRUCTION

A comprehensive instruction — Move Characters and Edit — is optionally available with the Model 200 processors and is a standard feature with the Type 1201, 1251, 2201, and 4201 processors. Processed information is edited before being converted to an output medium (e.g., a printed document) by the suppression of unwanted characters and symbols and the insertion of identifying symbols such as the dollar sign, decimal point, and asterisk. The Move Characters and Edit instruction is described on page 8-104.

ADDITIONAL READ/WRITE CHANNELS, UNIT LOADS, AND ADDRESS ASSIGNMENTS

As explained above, the number of peripheral operations that can be performed simultaneously by a processor depends on the number of read/write channels available, and the number of peripheral devices connectable to a processor depends on the number of unit loads and address assignments associated with the processor. Four optional features allow a user to increase his processor's peripheral flexibility by adding the following elements:

1. Feature 015 — Eight additional unit loads for a Model 200 processor. (The address assignments required to specify the additional peripheral controls enabled by this feature already exist in the basic 200 processors.)

2. Feature 016 — One additional (auxiliary) read/write channel for a Model 200 processor. (Auxiliary read/write channels are described on page 2-16.)

3. Feature 1115 — A second "I/O sector" for the Type 2201 processor.[1] This sector consists of four additional read/write channels, 16 additional unit loads, and 16 additional address assignments, thereby matching the peripheral capabilities of the basic I/O sector.

4. Feature 1116 — A third I/O sector for the Type 4201 processor.[1] (The basic 4201 processor contains two I/O sectors.) The optional sector consists of four additional read/write channels, 16 additional unit loads, and 16 additional address assignments. In addition to the third sector, Feature 1116 includes two additional read/write channels to be used with sector 1 and two additional channels to be used with sector 2. Feature 116 is described in Appendix H.

5. Feature 4214A and 4214B — Provides the Type 4201 processor with buffered I/O sectors for those applications where additional computer time or a higher input/output transfer capability is required. Feature 4214A requires the installation of Feature 1116, and Feature 4214B requires the installation of Feature 4214A. These features are discussed in Appendix H.

[1] An I/O sector consists of three elements: 4 read/write channels, 16 unit loads, and 16 address assignments.

6. Feature 4215 — A high-speed third sector for the Type 4201 processor. This allows connection of I/O peripheral devices with transfer rates exceeding 500,000 characters per second to the third sector. Feature 4215 requires the installation of Feature 1116. These features are discussed in Appendix H.

STORAGE PROTECT

Two Storage Protect Features, identical in nature, are offered for the Type 1201/1251, and 2201 processors as Features 1114 and 1117, respectively. These features allow a programmer-specified portion of the main memory (and the contents thereof) to be shielded from accidental alteration by programs running concurrently in the memory. An attempt to violate the protection of this area results in an "internal" processor interruption. The program or programs running in the protected memory area have 15 additional index registers at their disposal; these registers can also be used by programs in the unprotected (or "open") memory area if desired. The Storage Protect Features are described in Appendix E.

EXTENDED MULTIPROGRAMMING AND 8-BIT TRANSFER

The processing capabilities of the Models 1200, 1250, 2200, and 4200 are greatly extended by the addition of the Extended Multiprogramming and 8-Bit Transfer Features. These amplified capabilities are available as Features 1120, 1121, and 1118 for the Models 1200/1250, 2200, and 4200, respectively. Features 1120 and 1121 require that the Models 1200/1250, and 2200 be equipped with Storage Protect (Features 1114 and 1117, respectively); the Storage Protect capabilities are automatically included in Feature 1118 for the Model 4200. In addition to the capabilities supplied by the Storage Protect Features, extended multiprogramming provides storage protection with memory address relocation, interrupt masking, and instruction timeout. The 8-bit transfer capability gives the Models 1200, 1250, 2200, and 4200 increased flexibility by allowing either 8-bit or 6-bit information transfers between certain peripheral controls and main memory. The Extended Multiprogramming and 8-Bit Transfer Features are described in detail in Appendix G.

SCIENTIFIC UNIT

The scientific unit adds 14 scientifically oriented instructions to the Series 200 repertoire. The two functionally identical units — Feature 1100A (available with the Type 1201, 1251, and 2201 processors) and Feature 1101 (for the Type 4201) — are summarized in Appendix F.

FEATURE 0191

Feature 0191, which is available on the 1201, 1251, and 2201 processors and standard on the 4201 processor, enhances the instruction repertoire of the processor and affords increased compatibility with competitive equipment. This feature provides two additional instructions — Move or Scan (MOS) and Table Lookup (TLU) — and also includes the "S" (Special) mode of processing. The "S" mode of processing, which is implemented by the variant character of the Change Addressing Mode (CAM) instruction, enables the processor to manipulate the Add, Subtract, Zero Add, Zero Subtract, and Branch if Character Equal instructions in a special way. These instructions are described in Section VIII of this manual. The Move or Scan and Table Lookup Instructions, which are assembled by Easycoder Assembler D, are also discussed in Section VIII.

2

THE CENTRAL PROCESSOR

A Series 200 central processor is logically divided into five basic units (see Figure 2-1): a main memory, a control memory, an arithmetic unit, a control unit, and an input/output traffic control.

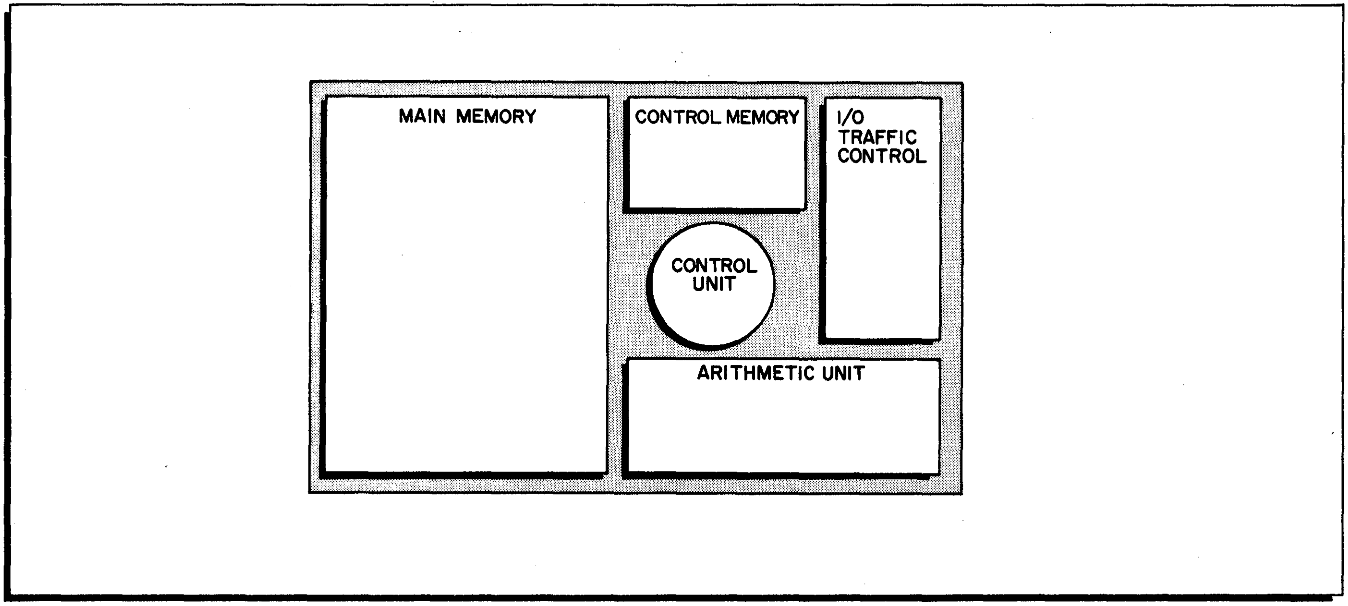


Figure 2-1. Logical Division of Series 200 Central Processor

MAIN MEMORY

The main memory contains from 2, 048 to 524, 288 character locations of magnetic core storage which are used to store program instructions and data during a program run (see Figure 2-2). Every character location is identified by a unique numeric address. This means that an instruction can designate the exact storage locations that contain the data needed for a particular operation.

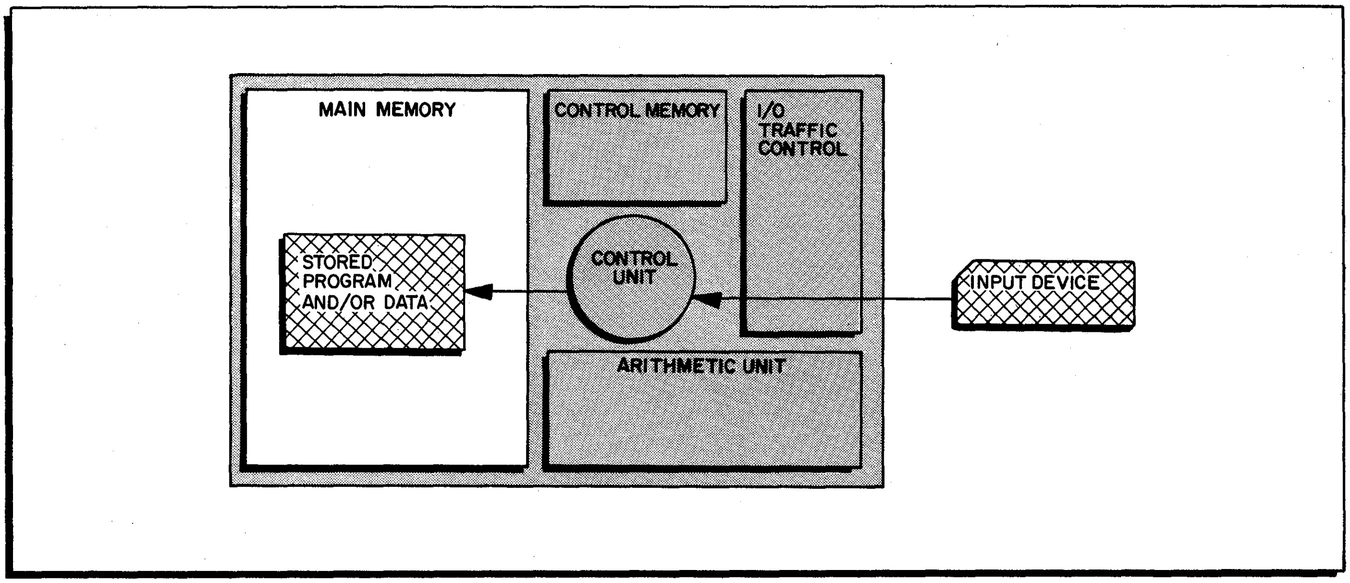


Figure 2-2. Main Memory Functions

Figure 2-3 shows one character position of memory with the name of each core shown to the right. Each core can be individually magnetized to represent either a one or a zero, depending upon its polarity. Moving from bottom to top in Figure 2-3, the first six cores are used for data storage, the seventh and eighth cores are used to define the limits of storage areas (these two cores are frequently referred to as "punctuation" bits), and the ninth core is used for parity checking.

Figure 2-4 shows how typical numeric, alphabetic, and special characters are stored in the main memory. Shaded circles represent cores containing 1-bits. Bits 1, 2, 4, and 8 in each character position can be combined to represent the decimal values zero through nine. This four-bit representation of decimal numbers is known as binary-coded decimal (BCD). Alphabetic and special characters are represented by a combination of the numeric (1, 2, 4, and 8) and the A and B cores. The A and B cores correspond to zone punches on cards: the A bit represents a 12-punch, the B bit represents an 11-punch, a combination of the A and B bits represents a 0-punch. A listing of the main memory formats for all valid Series 200 characters appears in Appendix B.

The word-mark bit (WM) is used to define logical storage fields in the memory. Information is rarely stored in the memory as single, independent characters; instead, adjacent character positions are usually grouped to form storage fields. As described in Section III, the word-mark bit is instrumental in defining the size of such fields.

CORE	FUNCTION			
◎	PARITY BIT (P)			
◎	ITEM-MARK BIT (IM)		PUNCTUATION BITS	
◎	WORD-MARK BIT (WM)			
◎	B BIT	ZONE BITS		DATA BITS
◎	A BIT			
◎	8 BIT	NUMERIC BITS		
◎	4 BIT			
◎	2 BIT			
◎	I BIT			

Figure 2-3. One Memory Position

		CHARACTER							
		Ø	4	9	B	M	,	(	F
BIT CONFIGURATION	P	●	○	●	●	●	○	●	○
	IM	○	○	○	○	○	○	○	○
	WM	○	○	○	○	○	○	○	○
	B	○	○	○	○	●	●	●	○
	A	○	○	○	●	○	●	●	●
	8	○	○	●	○	○	●	●	○
	4	○	●	○	○	●	○	●	●
	2	○	○	○	●	○	●	○	●
	I	○	○	●	○	○	●	○	○

Figure 2-4. Representation of Characters in Magnetic Core Storage

Consecutive storage fields are frequently grouped together to form a unit of information called an item. As its name implies, the item-mark bit (IM) is used to define the size of an item in the main memory (see Section III).

A unit of information that is to be transferred between the main memory and a peripheral device is called a record. A record can be of any length, from one character up to virtually the maximum number of characters in the memory. Both the word-mark and item-mark bits are used in defining the size of a record (see Section III).

The parity bit (P) is used in conjunction with an automatic error-detection technique known as parity checking. Every character must be represented in the central processor by an odd number of 1-bits. (Punctuation bits are excluded from this rule except in the Type 4201.) Whenever a character is moved from one location to another it is automatically checked to determine if an odd number of data 1-bits has been moved. In Figure 2-4, the characters 0, 9, B, M, and (are represented by an even number of ones in the data bit positions. Circuitry within the central processor automatically adds a one in the parity bit positions of these characters to provide the required odd bit count.

MEMORY CYCLE

The time interval required by a processor to read or write the contents of a unit location is termed memory cycle time. For the processors described in this manual, memory cycle time ranges from 2 microseconds (Model 200) down to 750 nanoseconds (Model 4200).

MAIN MEMORY IN THE TYPE 4201 PROCESSOR

The main memory of the Type 4201 processor consists of from one to four modules of core memory and a memory controller (see Figure 2-5). Each module is four characters in width and either 16,384 or 32,768 four-character groups in length. Thus, a module contains either 65,536 or 131,072 characters. Data storage capacities of main memory range from 131,072 to 524,288 characters.

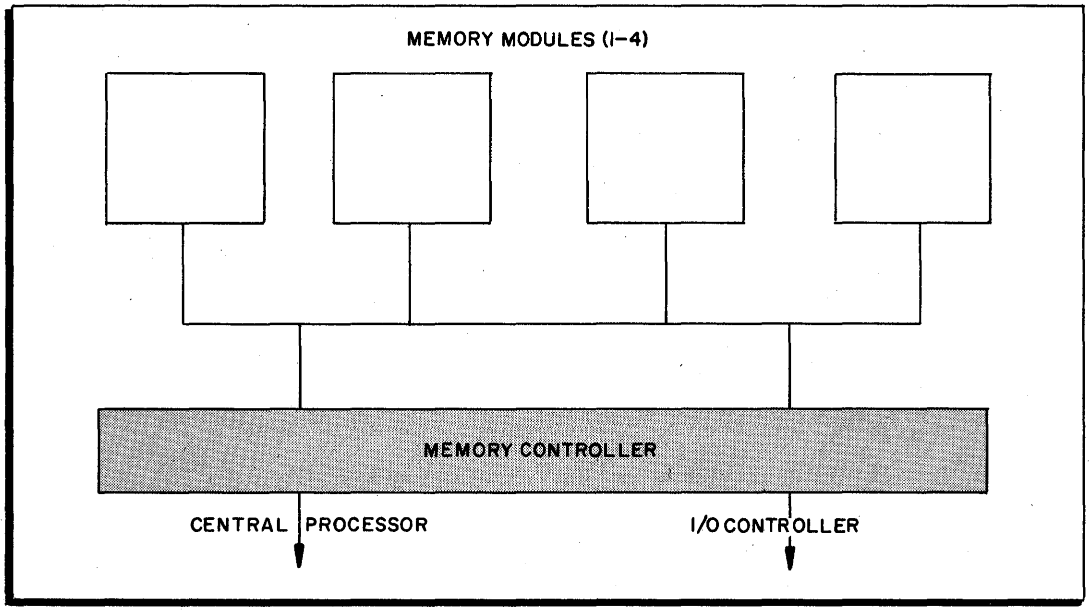


Figure 2-5. Type 4201 Memory Subsystem

Table 2-1 below shows the memory configurations available with the Type 4201 processors.

Table 2-1. Memory Configurations for Type 4201 Processors

Type	Memory Size in Characters
4201-3	131,072
4201-4	196,608
4201-5	262,144
4201-5A* (2-way interleaving)	262,144
4201-6	327,680
4201-7	393,216
4201-8	458,752
4201-9* (4-way interleaving)	524,288

*Memory addresses are interleaved across modules in these processors.

Memory Access

The 4201 processor always reads or writes the contents of four character locations at a time; such a four-character grouping is termed a "word." Thus, the Model 4200 has an effective memory cycle time of 750/4 or 188 nanoseconds per character.

Processing Unit

Although the 4201 processor always reads or writes a four-character word every memory cycle, the portion of the accessed word actually available for processing, called a "processing unit," varies from one to four characters, depending upon the operation being performed. The processing unit for a move instruction is up to four characters, whereas arithmetic and I/O instructions process one character at a time.

Memory Controller

The memory controller provides maximum simultaneity of memory operations by its ability to transfer data to or from memory modules simultaneously. This is accomplished by providing a set of read/write electronics for each memory module, so that access can be made to a module independently of all other modules. This ability allows internal processing and input/output operations to proceed independently and simultaneously. Simultaneous access occurs as long as the central processor and the I/O controller request access to different modules of memory. Whenever their requests are for the same module, the memory controller resolves the conflict by giving priority to the input/output controller.

When memory is addressed, a 4-character group containing the addressed character is delivered to either the central processor or the I/O controller. The delivery of four characters serves to significantly reduce the number of memory references for many operations and greatly increases the operating speed of the system.

Interleaved Addressing

In order to achieve optimum utilization of memory, an interleaved addressing scheme has been incorporated in two Model 4200 central processors (Types 4201-5A and -9). The use of the interleaved memory permits faster program execution by allowing multiple access to separate modules of memory to proceed simultaneously. This method of addressing is accomplished by assigning successive addresses to different modules so that a program written in a normal sequential manner will address different modules as it proceeds. For example, in the Type 4201-9 Central Processor, there are four memory modules which permit 4-way interleaving of accesses. With four modules, addresses 0, 1, 2, 3 are assigned to the first module; 4, 5, 6, 7 are assigned to the second module, etc. (see Figure 2-6).

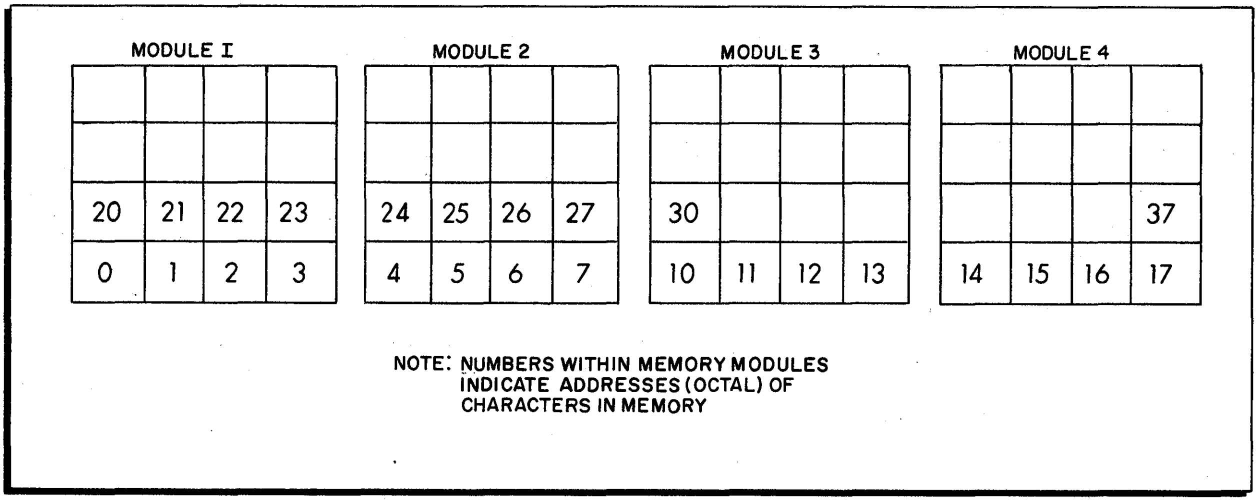


Figure 2-6. Model 4200 Memory Interleaving (Type 4201-9 Central Processor)

In addition, interleaved addressing further increases system performance by allowing the central processor to overlap many of its memory operations. This is accomplished by a unique design of the addressing circuitry of the Model 4200. Although 750 nanoseconds are required to cycle main memory, the addressing and data path circuitry of the processor is used only for a portion of the cycle time (approximately 500 nanoseconds). Therefore, the addressing circuitry is available for another memory access before the first access is completed. By interleaving, the instructions and operands will have been distributed among the available modules; therefore, the central processor is able to overlap successive fetches of both operands and the characters within an instruction.

Parity Check

Unlike the other Series 200 processors, the 4201 includes the punctuation bits in its parity check. Whenever a character is moved from one location to another, it is automatically checked to determine if an odd number of 1-bits in the data and punctuation positions has been moved.

CONTROL MEMORY

The control memory is a high-speed storage unit consisting of up to 57 control registers. (The number of registers actually available depends on the system configuration.) Normally, control registers contain the addresses of instructions and data being processed during a program run. One such register, called the A-address register, is illustrated in Figure 2-7. In this example, the A-address register contains an address (206) designating a main memory location, which in turn contains a unit of information (the decimal digit 7) to be added in the arithmetic unit.

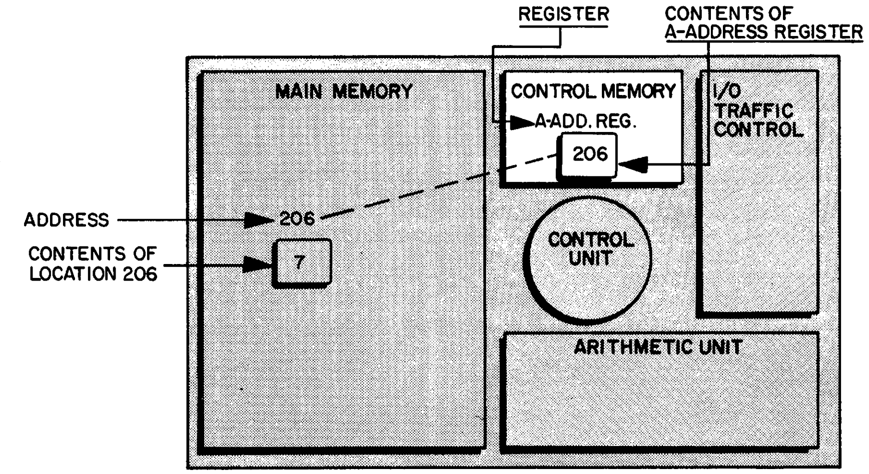


Figure 2-7. Typical Control Register Function

In Series 200 processors, other than the Type 4201, that do not include the Scientific Unit (Feature 1100A or 1101), each control register is only as large as it need be to contain the largest, or "highest," main memory address in the user's processor. Thus, a processor whose main memory capacity is 8,192 characters contains control memory registers which are each 13 bits long (13 bits allows 8,192 addresses), while the control registers of a processor containing 131,072 characters of memory storage are each 17 bits long (see Table 2-2). In a Type 4201 processor, all 19 control register bits are active, regardless of main memory size. When the Scientific Unit is included in the system, each control register is 18 bits (three characters) long (or 19 bits in the case of the Type 4201).

Table 2-2. Size of Control Memory Registers (Models 200/1200/1250/2200/4200)

MAIN MEMORY CAPACITY (Characters)	4,096	8,192	16,384	32,768	65,536	131,072	262,144	524,288
SIZE OF CONTROL MEMORY REGISTER (Bits)	12	13	14	15	16	17	18	24*
*19 address bits and 5 parity bits.								

Control registers can be addressed either by programmed instruction or from the operator's control panel or console. For instance, an instruction can change the course of a program by manipulating the contents of the control register that governs program sequence; the operator can interrogate a control register to determine the exact location at which the program has halted, etc. When a register is addressed by programmed instruction, it is specified by means of a variant character in the instruction. A register is addressed from the control panel or console by using the register's octal address. The functional name of each register and the variant character which specifies the register are listed in Table 2-3.

ADDRESS REGISTERS

The A- and B-address registers, the two sequence registers, and the interrupt registers are used to address main memory during the loading and execution of instructions. A detailed description of these registers is presented in Section IV, "Addressing."

READ/WRITE COUNTERS

Data is transferred between the main memory and a peripheral device via a read/write channel (described in Section I). Associated with a read/write channel are two location counters: a starting location counter and a current location counter. When a peripheral transfer is to be performed, the address at which the transfer is to begin is stored in both counters. Then, as each successive character is transferred, the contents of the current location counter are incremented by one so that when the transfer is completed, this counter contains the address of the character position immediately following the position that terminated the transfer, i.e., one beyond the record-marked location (see Section III).

The availability of the starting and current addresses associated with an input/output area greatly simplifies the manipulation of variable-length records.

Table 2-3. Control Memory Registers

MNEMONIC DESIGNATION	FUNCTION	VARIANT CHARACTER
REGISTERS STANDARD IN ALL PROCESSORS		
AAR	A-Address Register	67
BAR	B-Address Register	70
SR	Sequence Register	77
CLC1	Read/Write Channel 1 — Current Location Counter	01
CLC2	Read/Write Channel 2 — Current Location Counter	02
CLC3	Read/Write Channel 3 — Current Location Counter	03
SLC1	Read/Write Channel 1 — Starting Location Counter	11
SLC2	Read/Write Channel 2 — Starting Location Counter	12
SLC3	Read/Write Channel 3 — Starting Location Counter	13
WR1	Work Register 1[1]	-
WR2	Work Register 2[1]	-
WR3	Work Register 3[1]	-

Table 2-3 (cont). Control Memory Registers

MNEMONIC DESIGNATION	FUNCTION	VARIANT CHARACTER
FEATURE 010 or 011 (ADVANCED PROGRAMMING)		
CSR	Change Sequence Register	64
FEATURE 012 (PROGRAM INTERRUPT)		
EIR	External Interrupt Register	66
FEATURE 016 (AUXILIARY RWC)		
CLC1'	Read/Write Channel 1' - Current Location Counter	05
SLC1'	Read/Write Channel 1' - Starting Location Counter	15
FEATURES 1100A & 1101 (SCIENTIFIC UNITS)		
AC0	Floating-Point Accumulator 0[2]	- - -
AC1	Floating-Point Accumulator 1[2]	- - -
AC2	Floating-Point Accumulator 2[2]	- - -
AC3	Floating-Point Accumulator 3[2]	-
REGISTERS STANDARD ON 4201, OTHERWISE NOT AVAILABLE		
WR4	Work Register 4[1]	-
WR5	Work Register 5[1]	-
WR6	Work Register 6[1]	-
WR7	Work Register 7[1]	-
FEATURES 1114, 1117, & 1118 (STORAGE PROTECTION)		
IIR	Internal Interrupt Register	76
FEATURE 1115 (SECOND INPUT/OUTPUT SECTOR) - (STANDARD ON 1251 AND 4201)		
CLC4	Read/Write Channel 4 - Current Location Counter	21
CLC5	Read/Write Channel 5 - Current Location Counter	22
CLC6	Read/Write Channel 6 - Current Location Counter	23
CLC4'	Read/Write Channel 4' - Current Location Counter	25
SLC4	Read/Write Channel 4 - Starting Location Counter	31
SLC5	Read/Write Channel 5 - Starting Location Counter	32
SLC6	Read/Write Channel 6 - Starting Location Counter	33
SLC4'	Read/Write Channel 4' - Starting Location Counter	35
FEATURE 1116 (THIRD INPUT/OUTPUT SECTOR FOR 4201)		
CLC5'	Read/Write Channel 5' - Current Location Counter	26
CLC6'	Read/Write Channel 6' - Current Location Counter	27
SLC5'	Read/Write Channel 5' - Starting Location Counter	36
SLC6'	Read/Write Channel 6' - Starting Location Counter	37

Table 2-3 (cont). Control Memory Registers

MNEMONIC DESIGNATION	FUNCTION	VARIANT CHARACTER
CLC8	Read/Write Channel 8 - Current Location Counter	00
CLC9	Read/Write Channel 9 - Current Location Counter	20
SLC8	Read/Write Channel 8 - Starting Location Counter	10
SLC9	Read/Write Channel 9 - Starting Location Counter	30
CLC8'	Read/Write Channel 8' - Current Location Counter	04
CLC9'	Read/Write Channel 9' - Current Location Counter	24
SLC8'	Read/Write Channel 8' - Starting Location Counter	14
SLC9'	Read/Write Channel 9' - Starting Location Counter	34
CLC2'	Read/Write Channel 2' - Current Location Counter	06
CLC3'	Read/Write Channel 3' - Current Location Counter	07
SLC2'	Read/Write Channel 2' - Starting Location Counter	16
SLC3'	Read/Write Channel 3' - Starting Location Counter	17

[1] These registers are available only to the processor and must not be addressed by the program.

[2] These registers (accumulators) can only be addressed by the instructions included in Features 1100A or 1101 (see Appendix F).

ARITHMETIC UNIT

Arithmetic and logical operations are performed by a configuration of components commonly referred to as the arithmetic unit. Basically, this unit is composed of an adder, capable of performing both binary and decimal arithmetic, and two operand storage registers.[1] Each one of these units is capable of storing a single six-bit character in processors smaller than the Type 4201. The adder and operand storage registers in the 4201 processor can store four characters at a time.[2] In general terms, an arithmetic or logic operation is performed as follows (see Figure 2-8):

1. An instruction in the stored program specifies the type of operation to be performed and the main memory storage locations of the data to be manipulated.
2. The operands are transferred to the operand storage registers a character (Models 200, 1200, 1250, and 2200) or a word (Model 4200) at a time, beginning with the rightmost character in each operand.
3. In processors other than the 4201, each pair of characters (or, in the Model 4200, each pair of words) that enters the storage registers is combined in the adder. The result is stored in the main memory as specified by the program instruction. If a carry is generated, it is stored in the adder and combined with the next higher-order pair of characters.
4. In the 4201 processor, the storage registers and adder are used in the same manner as in other processors, except when performing address indexing or

[1] The contents of these registers are not accessible to the programmer.

[2] When floating point is installed in the 4201, the adder and operand storage registers are extended to a 6-character width in order to handle floating point operands.

floating-point operations.[1] That is, operations other than these two types are performed on a character-by-character basis. However, address indexing and floating-point operations take advantage of the full width of the 4201 adder.

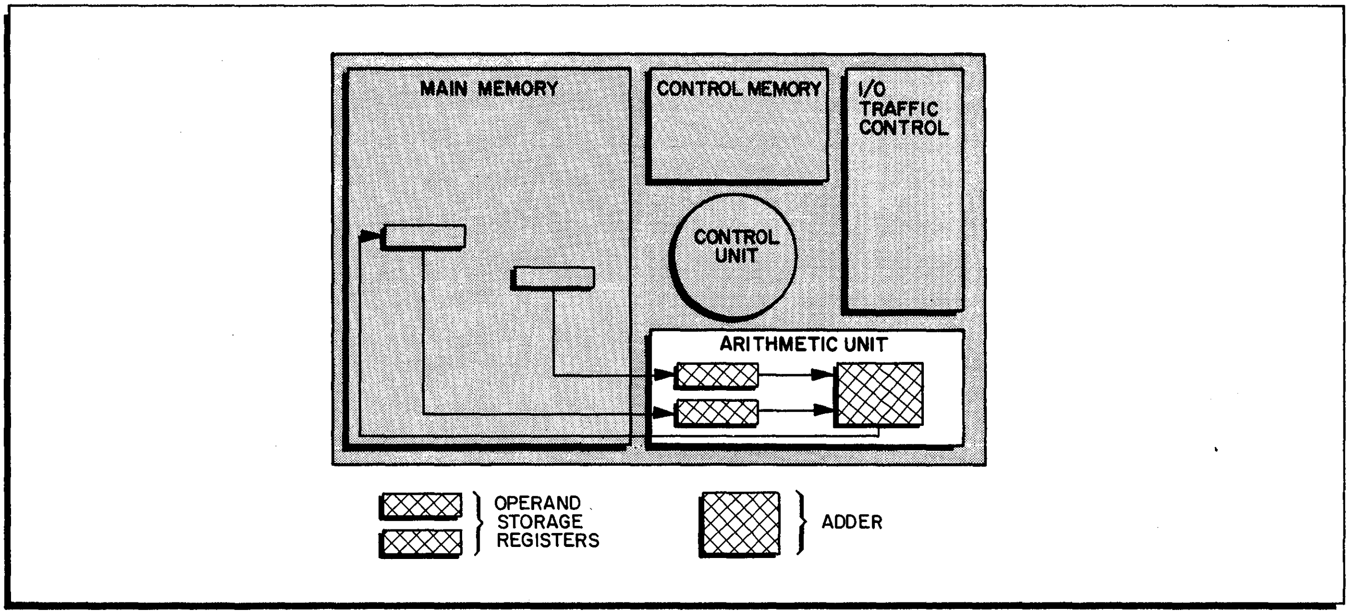


Figure 2-8. Data Flow Between Main Memory and Arithmetic Unit

CONTROL UNIT

The control unit is the hub of central processor activities (see Figure 2-9). Its major function is to select, interpret, and execute all of the instructions in the stored program. In carrying out these instructions, the control unit coordinates the various activities of receiving data from input devices, transferring data within the central processor, and transferring processed data to the output units. The main memory addresses used by the control unit in performing these tasks are stored in the registers of the control memory.

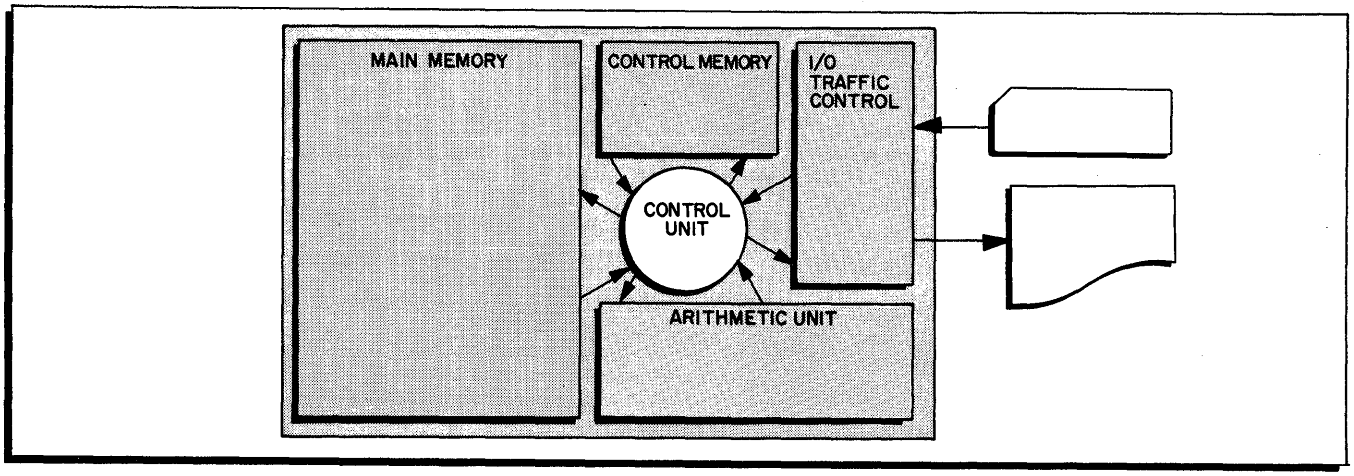


Figure 2-9. Control Unit Activities

[1] When floating point is installed in the 4201, the adder and operand storage registers are extended to a 6-character width in order to handle floating point operands.

INPUT/OUTPUT TRAFFIC CONTROL

The input/output traffic control is, as its name implies, the unit which regulates the flow (or "traffic") of data transferred during input/output activities. It works in conjunction with the central processor control unit to allocate central processor time to input/output operations and to identify the peripheral controls which are to use that time to transfer data (see Figure 2-10).

The I/O traffic control enables from 3 (Model 200 minimum) to 16 (Model 4200 maximum) simultaneous input/output operations to occur concurrently with the internal computations of the processor. In processors other than the 4201, this simultaneity is achieved by the traffic control's allocation of consecutive memory cycles to either peripheral controls or the central processor. In the Type 4201 processor, such allocation is not normally necessary, as independent cycling of memory blocks allows absolute simultaneity between memory accesses for I/O and computing operations (see page 2-4). Only when I/O and computing operations attempt to gain access to the same memory block simultaneously does the 4201 allocate memory cycles between the two types of operations.

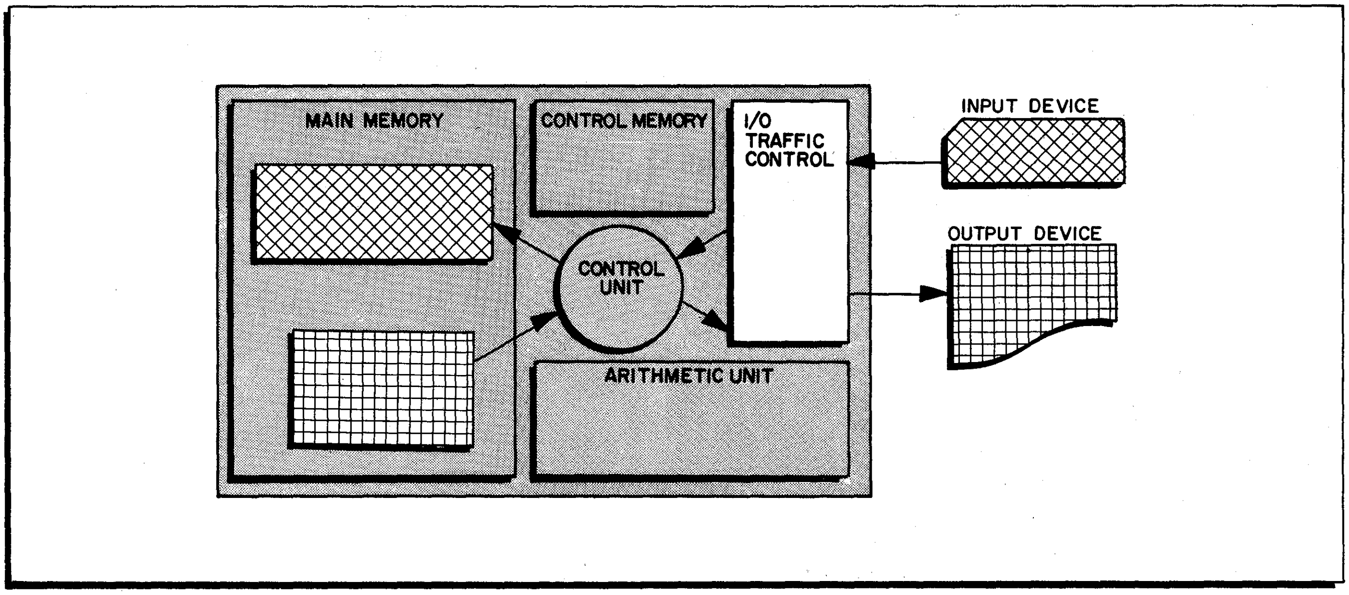


Figure 2-10. Input/Output Traffic Control Activities

MEMORY CYCLE DISTRIBUTION

Every peripheral data transfer involves some factor which prevents the device being used from transferring data at a rate comparable to that of the central processor. Usually this factor is mechanical — moving a card through the read station or a magnetic tape or disk past the read/write head — although in data communication it is the bit rate of the communication line. Therefore, a peripheral device requires access to the central processor to transfer information to or from the main memory during only a fraction of the time that the operation is proceeding. The periods in which the central processor is actually interrupted for data transfer are spaced over the duration of the peripheral operation (see Figure 2-11).

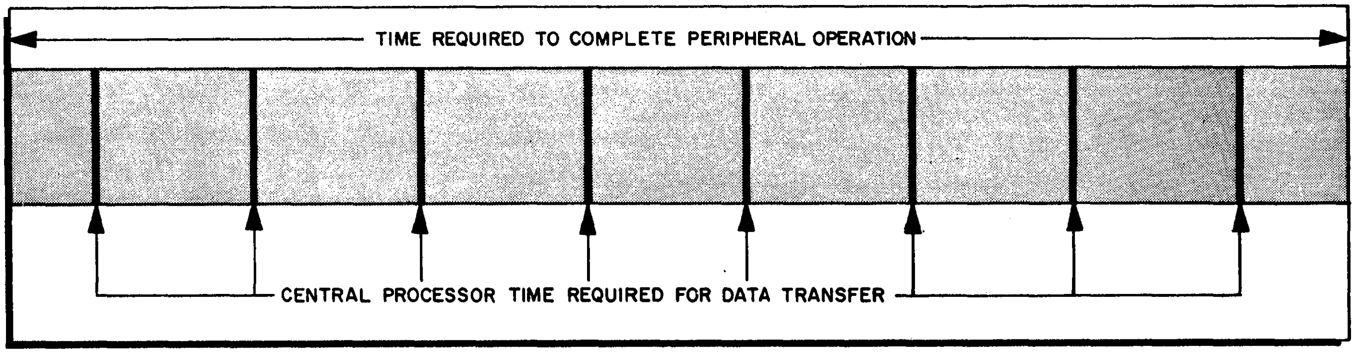

TIME REQUIRED TO COMPLETE PERIPHERAL OPERATION
CENTRAL PROCESSOR TIME REQUIRED FOR DATA TRANSFER

Figure 2-11. Data Transfer Intervals During One Peripheral Operation

When a peripheral operation is in progress but is not using main memory (the gray areas in Figure 2-11), another peripheral control may gain access to the main memory. This second memory access can in turn give way to a third access by another control before the original operation requires access to the memory again, etc. In other words, peripheral operations can occur simultaneously with one another. The periods of time in which peripheral controls do not require main memory access to transfer data are given to the central processor for its internal activities. It is the function of the I/O traffic control to direct the sharing of main memory cycles by the various peripheral devices and the central processor.

It was indicated on page 1-18 that in order for an I/O operation to proceed, the programmer must specify a read/write channel in the initiating peripheral instruction. This read/write channel completes the path between main memory and the control for the peripheral device being addressed. Input/output sectors (see page 1-22) consist of unit power loads, address assignments, and read/write channels. Type 1251 and 2201 processors may be equipped with two I/O sectors. Where this is the case, the read/write channel assigned to an operation must come from the sector to which the device being addressed is connected. Normally, this rule also applies to the 4201, which always has at least two sectors, but in that processor it is also possible to reassign RWC's outside of their "home" sectors by means of "sector escape codes" (see below).

The rate at which each peripheral control must transfer data over a programmer-assigned read/write channel(s) depends on the mechanical characteristics of the device connected to the control. Thus, the transfer intervals shown in Figure 2-11 are spaced according to the device being used. For instance, the transfer rate for the disk pack drive is considerably faster than that for the card punch; therefore, the disk pack drive will require access to the main memory more frequently than the card punch. The I/O traffic control monitors and honors the requests for access to the main memory. In processors other than the 4201, it decides how each memory cycle should be used — by a read/write channel or by the processor — as shown in Figure 2-13.

The traffic control offers consecutive memory cycles to read/write channels, one memory cycle per channel. If there is a demand on a particular channel when the cycle is offered, the channel is granted access to the main memory for one cycle. During this cycle a single character is transferred to or from memory. If the channel does not require the memory cycle, the cycle is given to the central processor for internal data processing.

In the Type 4201 processor, if an I/O operation requires access to the same memory block as the central processor, the I/O operation is given priority and the central processor stalls for one memory cycle. No interference (stall) occurs if the I/O operation and the central processor are accessing different memory blocks (i.e., memory accesses are simultaneous).

NOTE: In the Type 4201 processor, although a four-character word is moved in one memory cycle during internal processor operations, a single six-bit character is moved during input/output operations.

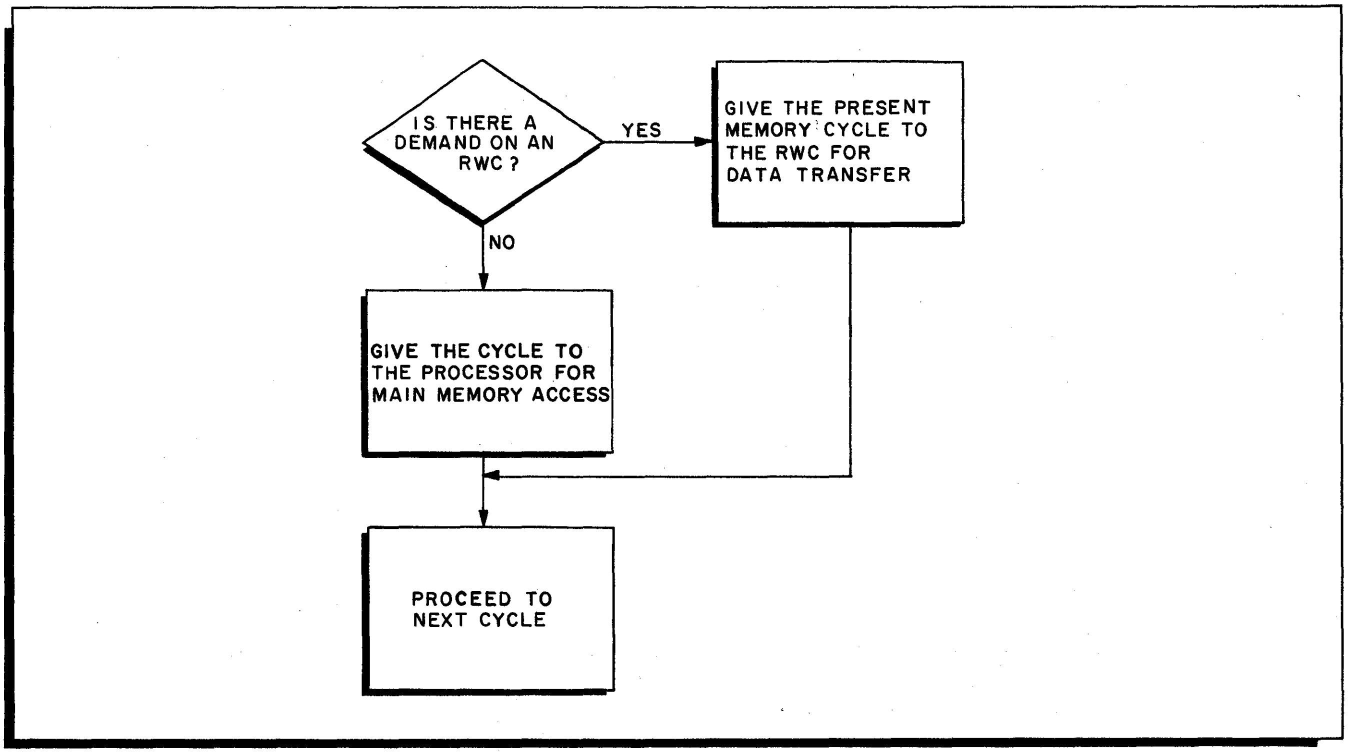
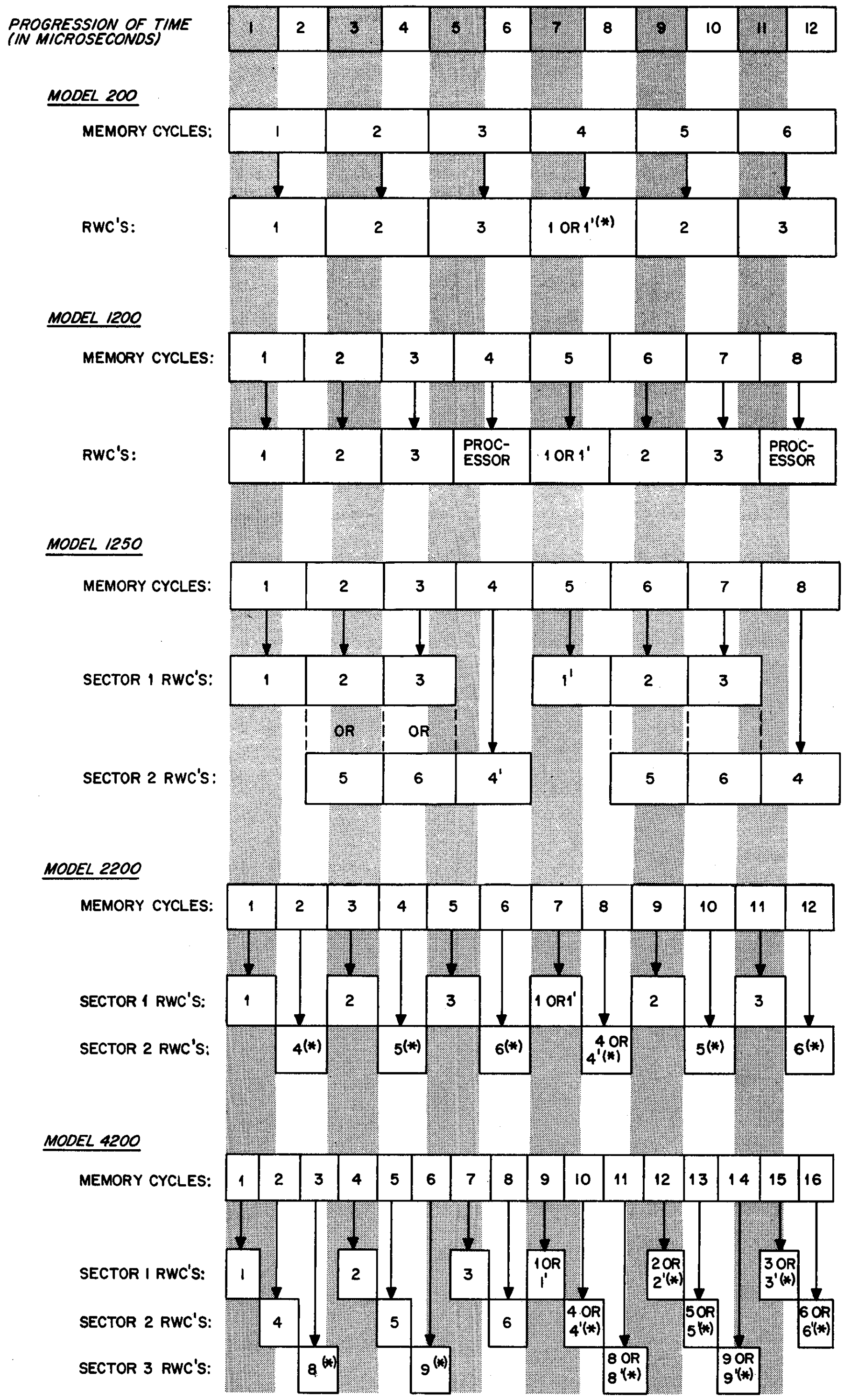

IS THERE A DEMAND ON AN RWC ?
YES
GIVE THE PRESENT MEMORY CYCLE TO THE RWC FOR DATA TRANSFER
NO
GIVE THE CYCLE TO THE PROCESSOR FOR MAIN MEMORY ACCESS
PROCEED TO NEXT CYCLE

Figure 2-12. Logical Decision Performed by Input/Output Traffic Control

The cyclic offering of memory cycles to read/write channels is shown in Figure 2-13. If the channel being offered a memory cycle is an optional channel (noted by an asterisk) that is not present in the user's system, the cycle is given unconditionally to the central processor.[1] Note that every fourth Model 1200 cycle is also given unconditionally to the processor. Note further that most channels available with the Models 200, 1200, 1250, and 2200 are offered main memory access once every six microseconds. Input/output speeds up to 167,000 characters per second can be

[1] There is one exception to this statement: if a Model 200 does not include RWC1' (Feature 016), the cycle is offered to RWC1.


PROGRESSION OF TIME (IN MICROSECONDS)
1 2 3 4 5 6 7 8 9 10 11 12
MODEL 200
MEMORY CYCLES: 1 2 3 4 5 6
RWC'S: 1 2 3 1 OR 1'(*) 2 3
MODEL 1200
MEMORY CYCLES: 1 2 3 4 5 6 7 8
RWC'S: 1 2 3 PROCESSOR 1 OR 1' 2 3 PROCESSOR
MODEL 1250
MEMORY CYCLES: 1 2 3 4 5 6 7 8
SECTOR 1 RWC'S: 1 2 3 1' 2 3
OR
OR
SECTOR 2 RWC'S: 5 6 4' 5 6 4
MODEL 2200
MEMORY CYCLES: 1 2 3 4 5 6 7 8 9 10 11 12
SECTOR 1 RWC'S: 1 2 3 1 OR 1' 2 3
SECTOR 2 RWC'S: 4(*) 5(*) 6(*) 4 OR 4'(*) 5(*) 6(*)
MODEL 4200
MEMORY CYCLES: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
SECTOR 1 RWC'S: 1 2 3 1 OR 1' 2 OR 2'(*) 3 OR 3'(*)
SECTOR 2 RWC'S: 4 5 6 4 OR 4'(*) 5 OR 5'(*) 6 OR 6'(*)
SECTOR 3 RWC'S: 8(*) 9(*) 8 OR 8'(*) 9 OR 9'(*)

(*)CHANNEL AVAILABLE AS AN OPTIONAL FEATURE.

Figure 2-13. Symbolic Representation of Input/Output Traffic Control[1]

[1] This figure is not applicable to 4201 operations employing channel transfer rates higher than the minimum.

maintained by accessing memory at these intervals. In processors other than 4201, transfer rates higher than those attainable with a single read/write channel can be achieved by interlocking two or more read/write channels, as described below.

Rather than interlocking RWC's, the Model 4200 traffic control offers variable numbers of memory cycles per unit of time to each read/write channel, depending upon the read/write channel assignment code used in the instruction which initiates the operation. From one to six cycles are offered to a read/write channel every 12 microseconds, giving channel data transfer capacities ranging from 83,300 to 500,000 characters per second. Effectively, then, the 4201 incorporates variable-speed read/write channels.

PRIMARY AND AUXILIARY READ/WRITE CHANNELS

RWC1', RWC2', RWC3', RWC4', RWC5', RWC6', RWC8', and RWC9' are called auxiliary read/write channels because of the manner in which they are granted access to the main memory by the traffic control. For instance, the Model 200 traffic control offers one cycle to RWC1, the next cycle to RWC2, the next cycle to RWC3, the next cycle to RWC1', the next cycle to RWC2, the next cycle to RWC3, the next cycle to RWC1, etc. In other words, memory cycle allocation alternates between a primary channel and its auxiliary channel.

Read/write channels not accompanied by auxiliary channels (e.g., RWC's 2 and 3 in the Model 200) are each guaranteed access to the main memory every six microseconds (giving a transfer rate of 167,000 characters per second), as shown in Figure 2-13. Primary channels and auxiliary channels are each granted access every 12 microseconds, because access is alternated between the two, thus providing a transfer rate of 83,300 characters per second.

INTERLOCKING READ/WRITE CHANNELS

As indicated above, in order to achieve data transfer rates higher than those attainable with a single read/write channel, it is necessary to interlock two or more read/write channels in processors other than the 4201. In this manner, data transfer rates from 167,000 to 500,000 characters per second are possible. The same instruction which initiates the data transfer operation specifies whether or not channels are to be interlocked. When this procedure is used, all of the cycles normally offered to the interlocked channels are made available to the single data transfer operation. The transfer rate thus provided is equal to the sum of the rates attainable individually with the interlocked channels. When the operation is completed, memory cycle allocation returns to normal and channels are again offered cycles at the normal intervals. Programming procedures for channel interlocking are described beginning on page 8-110.

MODEL 4200 VARIABLE-SPEED READ/WRITE CHANNELS

As indicated above, the 4201 is equipped with variable-speed RWC's. No more than two

RWC's (a primary and the corresponding auxiliary) are ever made busy by a single RWC assignment. However, a single RWC assignment can still command a data transfer capacity of up to 500,000 characters per second. The most important advantage of this arrangement is that the RWC's not made busy by a high-speed transfer are available for use in other operations. For example, in order to handle a 250,000-character-per-second I/O transfer, other processors would require the interlocking of several channels. In the 4200, only one primary channel will be tied up. The other RWC's in the same sector will still be available for use in other operations, e.g., three 83,300-character-per-second transfers. However, in no case can the total data transfer rate of a single sector exceed 500,000 characters per second.

Another feature of the Model 4200 — the "sector escape code" — makes variable-speed RWC's even more attractive. An escape code allows an RWC normally restricted to operating in one sector to be used for I/O data transfers in another sector. For example, an escape code can be used to assign RWC 1, normally used only in sector 1, to sector 2 I/O operation. Programming procedures for Model 4200 RWC's are described in Section VIII.

Table 2-4. Summary of Central Processor Characteristics

PROCESSOR	201	201-1	201-2	1201	1251	2201	4201
MAIN MEMORY							
PROCESSING UNIT	Six-bit character.						Four-character word.
INSTRUCTION FORMAT	Variable. Typical configuration: op code, two addresses, and variant character.						
ADDRESSING MODES	Two-, three-, and four-character addressing. Three- and four-character addresses can specify indexed and indirect addressing.						
MEMORY CAPACITY (Characters)	2,048-32,768	2,048-65,536	4,096-65,536	16,384-131,072	32,768-262,144	16,384-262,144	131,072-524,288
MEMORY CYCLE (microseconds)	2	2	2	1.5	1.5	1	.75/word (.188/character)
INDEX REGISTERS	0-6	0-15	0-15	15-30	15-30	15-30	15-30
CONTROL MEMORY							
MEMORY CAPACITY (Control Registers)	12-16	13-16	13-16	16-29	16-37	16-37	28-57
ACCESS TIME (Nanoseconds)	270	270	270	185	185	185	125
	ARITHMETIC UNIT						
OPERATIONS	Decimal arithmetic, binary arithmetic, logical operations.						
TYPICAL OPERATING SPEEDS (3-character address mode) 5-Digit Decimal Add (A+B→B)	48μs	48μs	48μs	35μs	35μs	25μs	13μs
5-Digit Compare (A:B)	38μs	38μs	38μs	29μs	29μs	21μs	10μs
CONTROL UNIT							
CHECKING	One parity bit with each character.						
PROGRAM CONTROL	Sequential selection, interpretation, and execution of all stored-program instructions.						
INPUT/OUTPUT TRAFFIC CONTROL							
READ/WRITE CHANNELS	3-4	3-4	3-4	4	8	4-8	8-16
ADDRESS ASSIGNMENTS	16	16	16	16	32	16-32	32-48
UNIT LOADS	8-16	8-16	8-16	16	32	16-32	32-48
SIMULTANEOUS OPERATIONS POSSIBLE	3-4	3-4	3-4	4	8	4-8	8-16

3
DATA FORMAT

VARIABLE FIELD LENGTH

Information is stored in the main memory in groups of characters, which are called fields. A field is, by definition, any group of characters that is treated as a unit. Series 200 computers permit fields of any length, from one character up to the maximum number of characters in the memory. This means that an instruction or data field occupies only that number of core storage locations actually needed.

The use of variable-length fields requires that there be a method of indicating the actual lengths of instruction fields and data fields. This requirement is fulfilled by the word-mark bit mentioned in Section 2. The word-mark bit performs the following functions:

1. It terminates the retrieval of an instruction.
2. It terminates the execution of an instruction.
3. It defines the size of a data field.

Throughout this manual, the presence of a word mark will be indicated by a circle around the character with which it is associated. The following points should be noted regarding the use of word marks:

1. Word marks can be set and cleared by programmed instructions.
2. Word marks are set by the same routine that loads a program and data into the main memory. Usually, word-mark assignments remain unchanged throughout the execution of a program.
3. An instruction is terminated by a word mark in the storage position immediately following its last (rightmost) character.
4. A data field is terminated by a word mark associated with its high-order (leftmost) character.[1]

[1] The footnote on page 3-4 describes an exception.

INSTRUCTION FORMAT

An instruction is a coded statement which orders the computer to perform a fundamental operation. A set of instructions suitably combined to perform a specific task is called a program or routine.

As will be shown in Section V, the task of coding the instructions in a program is greatly simplified by the use of the Easycoder symbolic programming system. The Easycoder Assembly Program converts the symbolic coding written by the programmer into a machine language which is acceptable to the internal logic of the machine.

OPERATION CODE

Basic to all instructions is an operation code, usually referred to as an op code, that defines the fundamental operation to be performed. The programmer specifies an op code by using a predefined mnemonic configuration; e.g., BA is the op code that specifies a "binary add" operation, MCW is the op code that specifies a "move characters to word mark" operation. The Easycoder Assembly Program automatically converts a mnemonic op code into a single-character, machine-language op code and sets the word-mark bit in the character position in which it is stored.

A AND B ADDRESSES

Most instructions also have two address portions, designated as the A address and the B address. The address portions indicate the starting locations of the operand fields in the main memory. Using the Easycoder language, the programmer can specify memory locations by means of symbolic addresses or "tags" (see Section V).

The Easycoder Assembly Program automatically assigns absolute memory addresses to the symbolic addresses appearing in a program (see Figure 3-1). Thus, the programmer can manipulate operands without regard to their actual storage locations in memory.

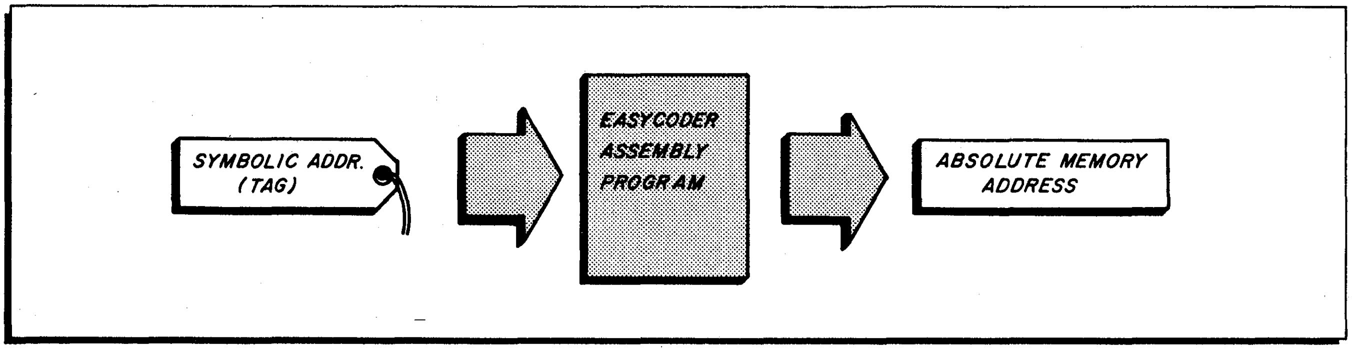


Figure 3-1. Conversion of Symbolic Tags to Absolute Memory Addresses

Because of the modular design of Series 200 computers, the programmer has the facility to specify whether a two-, three-, or four-character absolute address will be assigned to each symbolic address used in the program. In any case, the absolute addresses assigned by the assembly program are interpreted as pure binary numbers (see Section IV).

VARIANT CHARACTER

The variant character is used to modify the op code of an instruction. For example, the op code of a Branch on Condition Test instruction (BCT) specifies the fundamental operation "branch if a tested condition is met." The condition or restriction which must be met before the branch can occur is specified by the variant character. A table of valid variant characters is presented in Appendix B.

SUMMARY

Figure 3-2 shows the six basic formats in which machine-language instructions may appear. Since the maximum number of characters in an instruction depends upon whether two-, three-, or four-character addressing is being used, shaded boxes in the illustration indicate the format of an instruction without specifying the number of characters in each part. These formats are representative of all instructions except those associated with input/output and translate operations.[1] For the sake of direct comparisons, Figure 3-3 illustrates each of the formats defined in Figure 3-2 as a symbolic entry on the programmer's coding form.

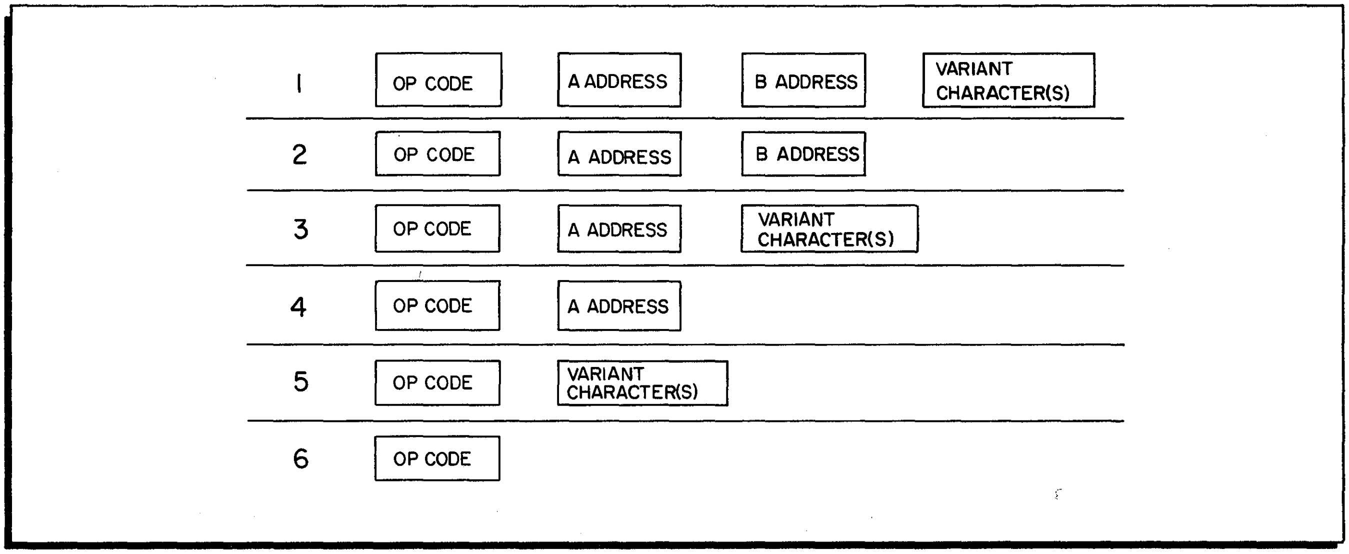
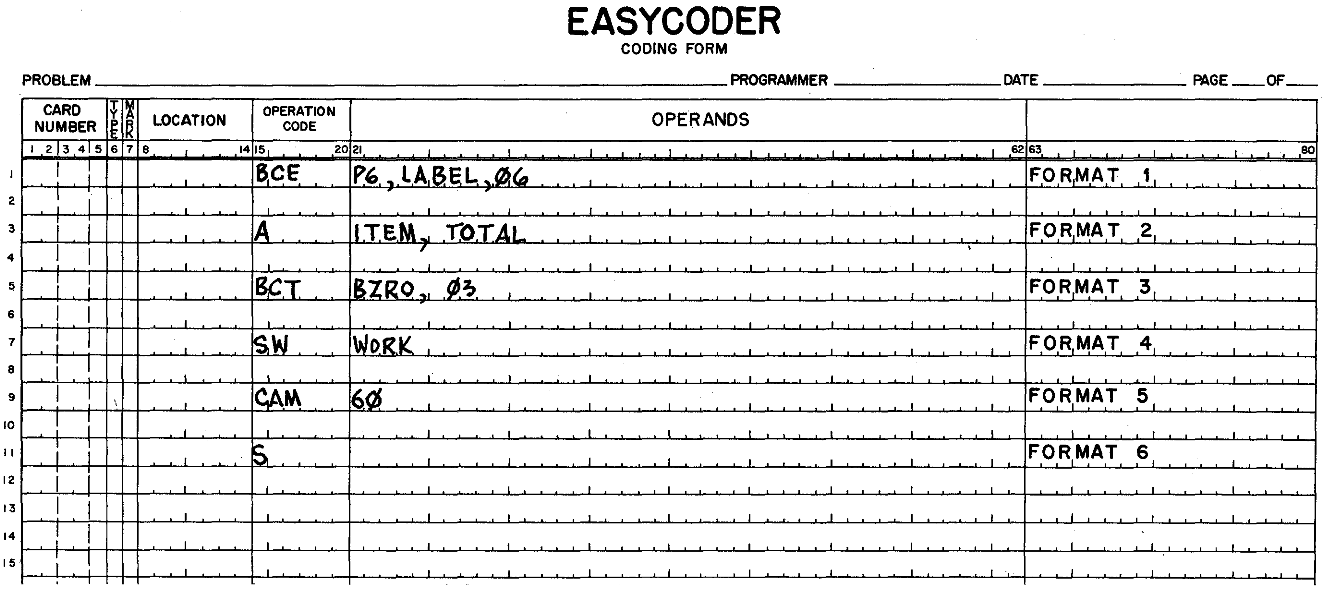


Figure 3-2. Series 200 Instruction Formats

[1] The format of an input/output instruction is a modification of format 3 shown in Figure 3-2. Specifically, the variant characters of the instruction are replaced by a field of one or more control characters which define the input/output operation in terms of data path, direction of data flow, control unit designation, etc. The format of a translate instruction is a modification of format 1 shown in Figure 3-2. In Section VIII, Series 200 instructions are described in terms of their individual formats.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					BCE	P6,LABEL,Ø6	FORMAT 1
2							
3					A	ITEM, TOTAL	FORMAT 2
4							
5					BCT	BZRO, Ø3	FORMAT 3
6							
7					SW	WORK	FORMAT 4
8							
9					CAM	6Ø	FORMAT 5
10							
11					S		FORMAT 6
12							
13							
14							
15							

Figure 3-3. Symbolic Representation of Series 200 Instructions

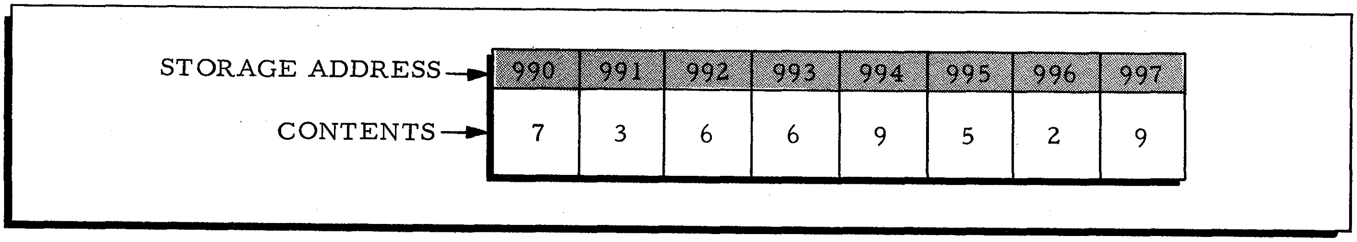
ORGANIZATION OF DATA IN MAIN MEMORY

Data may be stored in the main memory in any of the following variable-length formats:

- FIELD
- ITEM
- RECORD

FIELDS

Consider the eight consecutive storage locations shown in Figure 3-4. To indicate to the machine that these eight characters are to be treated as a field, their left and right boundaries must be defined. The left boundary is normally defined by setting a word mark in position 990. The right boundary is normally defined by specifying storage address 997 in the instruction that will manipulate the field.[1] The eight-character group shown in Figure 3-5 is thus defined as a field.



STORAGE ADDRESS →	990	991	992	993	994	995	996	997
CONTENTS →	7	3	6	6	9	5	2	9

Figure 3-4. Consecutive Storage Locations in Main Memory

[1] Although this is the conventional method of defining fields, the Extended Move (EXM) instruction (see Section VIII) permits a field to be defined by a word mark at either the left or the right boundary. The opposite boundary is then specified in the instruction.

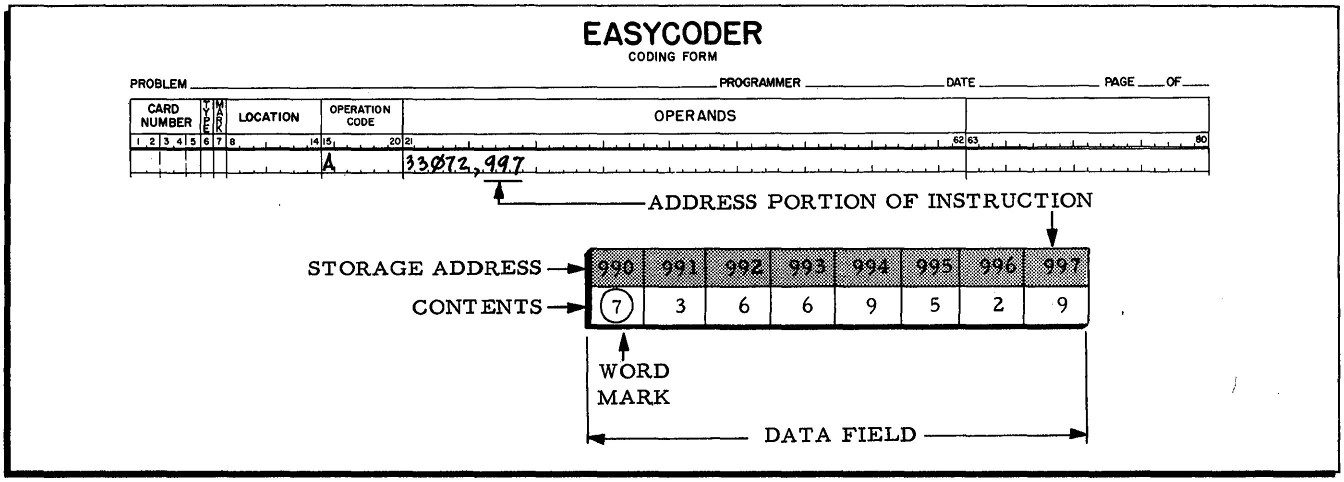


Figure 3-5. Data Field Format in Main Memory

ITEMS

An item consists of ,one or more consecutive storage locations whose boundaries can be defined in either of two ways:

1. The leftmost character position can be defined in the instruction that will operate on the item and the rightmost character position defined by an item mark; or
2. The rightmost character position can be defined in the instruction that will operate on the item and the leftmost character position defined by an item mark.

NOTE: An item mark is illustrated in this manual by underlining the character with which it is associated. Fields within an item are defined by word marks.

There are only two instructions that manipulate items — Move Item and Translate, and Extended Move. In the Move Item and Translate instruction, the leftmost character of an item is addressed and the rightmost character contains an item mark. In the Extended Move instruction, several different item boundaries can be specified by the variant character of the instruction.

Two items, each containing three data fields, are shown in Figure 3-6.

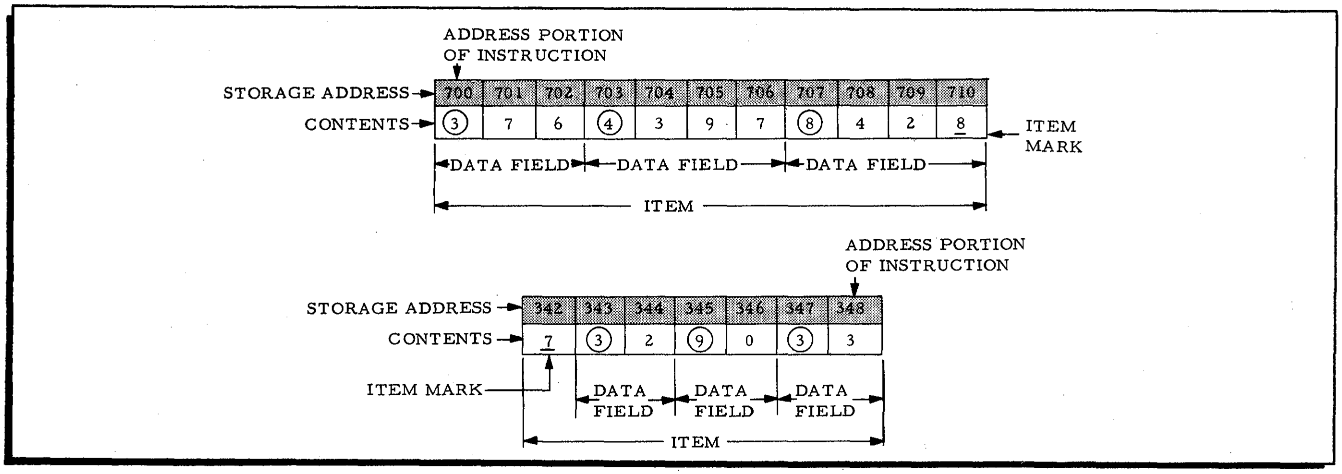


Figure 3-6. Two Item Formats in Main Memory

RECORDS

A record is any unit of information that is to be transferred between the main memory and a peripheral device. A record can be of any length, from one character up to the maximum number of characters in the memory. It can contain any number of items and fields. The rightmost limit of a record is defined by a record mark in the character position following the last character in the record (see Figure 3-7).[1]

NOTE: A record mark is illustrated by combining the word-mark and item-mark symbols. The address of the leftmost character in a record is specified in the instruction that operates on the record.[1]

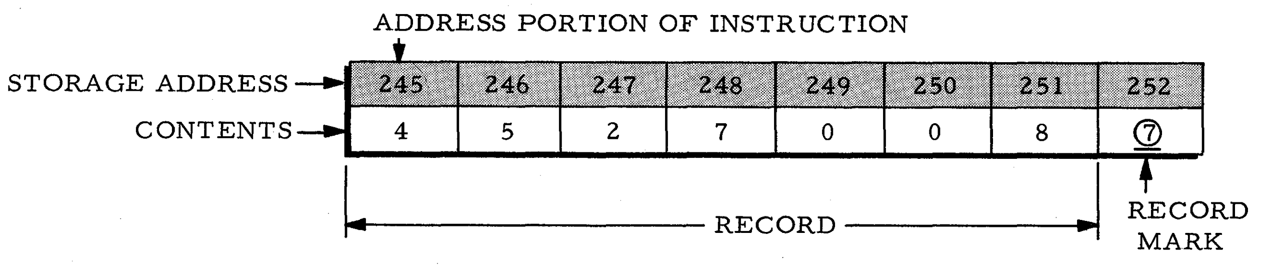


Figure 3-7. Record Format in Main Memory

SUMMARY

The foregoing data format conventions are summarized in Figure 3-8.

DATA FORMAT	BOUNDARY DEFINITION		INSTRUCTION USED TO SET MARK (See Section 8)
	LEFTMOST CHARACTER	RIGHTMOST CHARACTER	
FIELD	Word Mark ⊗	Address portion of instruction	Set Word Mark
ITEM	Address portion of instruction	Item mark $\underline{X}$	Set Item Mark
	Item Mark $\underline{X}$	Address portion of instruction	
RECORD	Address portion of instruction	Record mark $\underline{\otimes}$ (in character position following last character of record)[1]	BOTH Set Word Mark and Set Item Mark

Figure 3-8. Summary of Internal Data Formats

[1] A record can also be moved internally (i.e., from one main memory area to another) by means of the Extended Move instruction (see Section VIII). In this case, the character containing the record mark is considered as part of the record. This instruction can specify either the right or left boundary of the record to be moved.

MAGNETIC TAPE DATA FORMAT

In many applications, a major input and output medium for a Series 200 computer is magnetic tape. The standard Series 200 magnetic tape system uses 1/2-inch tape as the recording medium. A tape system using 3/4-inch tape is also available.

Information is stored on 1/2-inch magnetic tape in variable-length groups of characters called records. The tape is divided lengthwise into seven recording channels. A line of bit positions across the tape, one position for each channel, is called a frame. The seven bits in a frame correspond to the six information bits and one parity bit found in a character position in the main memory. Notice that no channels are provided for the storage of punctuation bits on tape. Unlike main memory records, which are delimited by record-mark punctuation, tape records are separated from each other by a band of blank tape, which is called an interrecord gap. The representation of a memory character position on magnetic tape is shown in Figure 3-9.

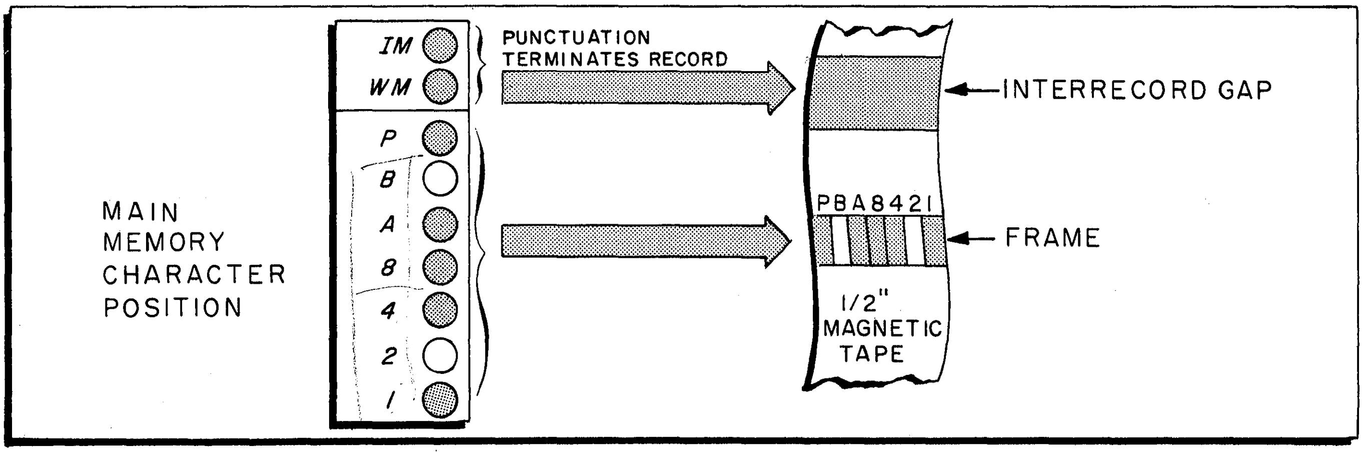


Figure 3-9. Character Representation on Magnetic Tape

Characters recorded on magnetic tape are transferred from the main memory without parity bits. At the time of recording, the magnetic tape control generates parity bits as required. The programmer may specify either odd- or even-parity recording: in the odd-parity mode the bit count in each frame is odd; in the even-parity mode the bit count is even.

In addition to parity bits, which are used for frame checking, the magnetic tape control also generates a longitudinal check frame which is used for channel checking purposes. A check frame is automatically appended at the end of each record stored on tape.

Recall that a record stored in memory is delimited by a record mark in the character position following the last character in the record. When a record is transferred to tape, the contents of the character position containing the record mark are not included as part of the record. On the other hand, if a record mark is sensed in memory when information is being read in from tape, the record mark will terminate the record and the character position containing the record mark will receive a character from the tape. Although data transfer from the tape is terminated by the record mark, tape motion continues until an interrecord gap is sensed. No punctuation marks are altered in any way as a result of tape read/write operations initiated by a program.

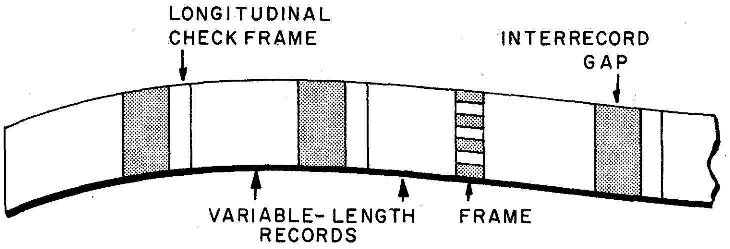


Figure 3-10. Data Format on Magnetic Tape

PUNCHED CARD FORMAT

Punched cards provide a convenient means of entering data into the machine. The cards used for this purpose are either standard 12-row, 80-column cards or 51-column cards. Each card column may contain a decimal digit, an alphabetic character, or a special symbol such as a slash or an asterisk (see Figure 3-11).

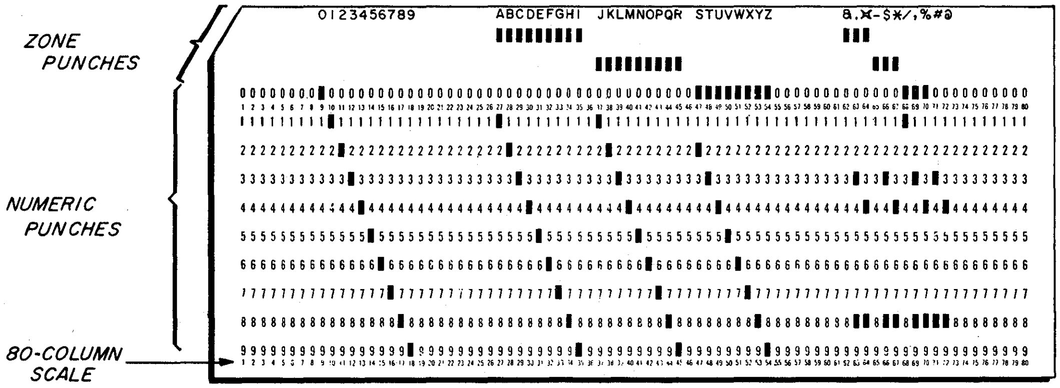


Figure 3-11. Punched Card Codes

Numeric information is represented using the card punch positions labeled zero through nine. Alphabetic information is represented by a combination of numeric punches and zone punches. There are three zone punch positions: the 12 zone at the top edge of the card, the 11 zone just below the 12-zone position, and the zero zone labeled as row zero on the card. The 11 and 12 zones are not labeled because the top edge of the card is reserved for printed headings.

In addition to Hollerith code, cards may be punched or read in the direct transcription mode as an optional feature. Each punch position on the card is individually significant in this mode, a punch representing a one bit and the absence of a punch representing a zero bit.

The data formats of the media most commonly associated with peripheral devices (viz., magnetic tape and punched cards) have been described. However, other media (e.g., paper tape, magnetic disks, etc.) also contain unique data formats which are converted to central processor format by their respective peripheral controls. These formats are described in the individual Series 200 publications which define such devices.

4

ADDRESSING

BASIC CONCEPTS

The main memory storage locations that contain the instructions and data of a program are identified to the machine by their particular main memory addresses. Every character storage location in the main memory is directly addressable.

An instruction is stored in a field of from 1 to 12 characters, depending on the format of the instruction and the mode of address assembly (two-, three-, or four-character). Figure 4-1 illustrates how a typical seven-character Add instruction appears when stored in the main memory. (Recall that enclosing a character in a circle indicates that a word mark is associated with it.)

An instruction is addressed by specifying the op code (leftmost) location of the instruction. For instance, the address of the Add instruction in Figure 4-1 is 524. The machine reads an instruction from left to right until it senses a word mark. For example, the extraction of the Add instruction (Figure 4-1) is stopped by the word mark associated with the op code of the next instruction in sequence.

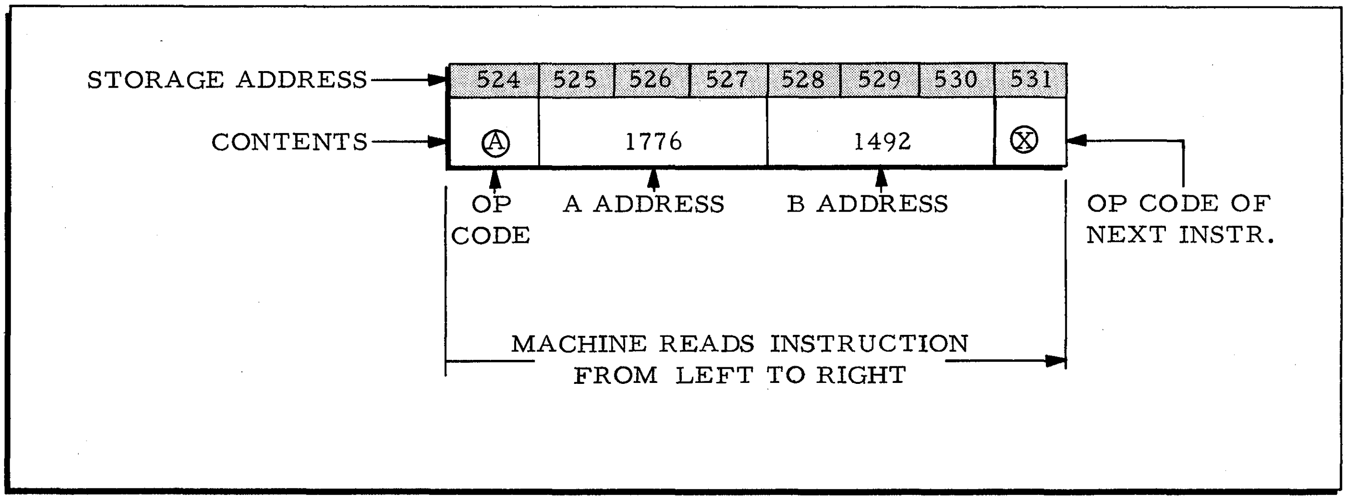


Figure 4-1. Typical Add Instruction

As mentioned in Section III, a data field is normally defined in the following manner: the leftmost location in the field is indicated by a word mark; the rightmost location is specified in the A or B address of an instruction. The machine reads a data field from right to left until it senses the word mark associated with the leftmost character in the field.[1] For example, the A and B addresses in the instruction shown in Figure 4-1 could specify the data fields shown in Figure 4-2.[2]

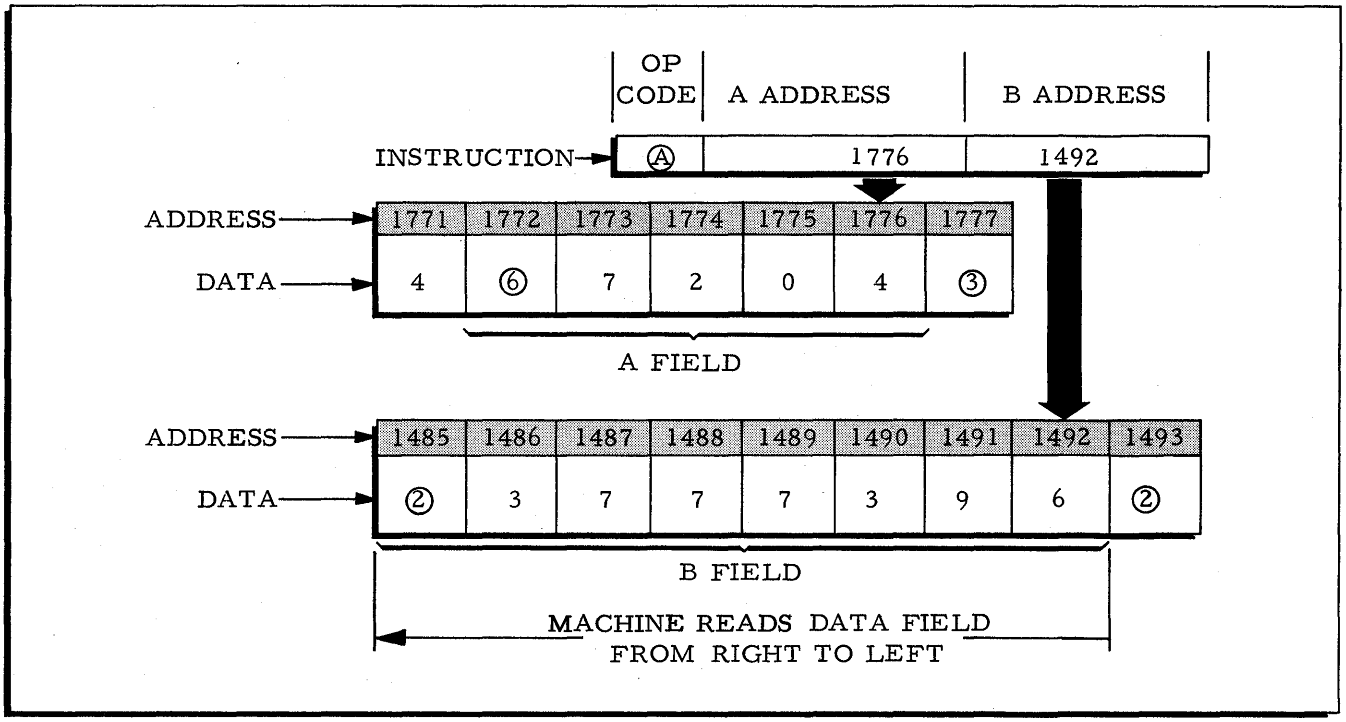


Figure 4-2. Extraction of Data Fields in Typical Add Instruction

An item is addressed by specifying either its leftmost or its rightmost character location in an address portion of an instruction (a variant character in the instruction specifies which character is being addressed). If the address of the leftmost character is specified, the machine reads the item from left to right; if the address of the rightmost character is specified, the machine reads the item from right to left. In either case, the operation terminates when an item mark is sensed.

A record is addressed by specifying its leftmost character location in an address portion of an instruction. The machine reads a record from left to right until it senses a record mark.[1] Note that the contents of the character position containing a record mark are not considered as part of the record, except when the record is moved internally.

[1] Recall that the Extended Move (EXM) instruction permits the reading of fields, items, and records in either direction.

[2] All examples and illustrations in this section are presented in decimal notation. A table of decimal and octal equivalents appears in Appendix A.

The direction in which the machine reads any of the above-mentioned groups is compatible with the manner in which the contents of the group are manipulated. For instance, a field used in an arithmetic operation is read from right to left because such operations combine fields character by character, starting with the low-order or "units" position in each field. Similarly, an instruction is read from left to right because the machine must interpret the op code before it can manipulate the operand(s).

REGISTERS USED IN ADDRESSING

The processing of a stored-program instruction consists of two phases: the retrieval (or "extraction") of the instruction from main memory storage, and the execution of the instruction. Six control memory registers are used to address the main memory during instruction processing. Four registers — SR, CSR, EIR, and IIR — are related to the sequential selection of instructions in a program; the other two registers — AAR and BAR — control the transfer of information from one storage location to another by containing the address portions of an instruction.

SEQUENCE REGISTER (SR)

SR contains the address of the next sequential instruction character to be extracted from the memory during a program run. The contents of SR are incremented by one as each instruction character is extracted, so that SR contains the address of the next instruction's op code when one instruction has been completely extracted.

CHANGE SEQUENCE REGISTER (CSR)

The address of an op code can be stored in CSR.[1] A Change Sequencing Mode instruction (see page 8-66) will interchange the contents of SR and CSR and thereby cause the program to branch to the instruction whose op code address was stored in CSR. At this point in the program CSR will contain the address of the op code following the Change Sequencing Mode instruction. In order to return to this op code (i.e., to the initial sequence of instructions), another Change Sequencing Mode instruction can be issued.

EXTERNAL INTERRUPT REGISTER (EIR)

EIR, like CSR, can be used to store the address of an op code.[1] This address and the contents of SR will be interchanged automatically when an external interrupt signal is received. (Recall that an external interrupt signal can be generated by a peripheral control, by the control panel or console, or by the Monitor Call instruction.) In order to return to the normal sequence of instructions that was interrupted, a Resume Normal mode instruction (see page 8-99) can be issued.

[1] A Load Control Registers instruction can be used to store the desired op code address (see page 8-60).

INTERNAL INTERRUPT REGISTER (IIR)

The address of an op code can also be stored in IIR.[1] When the Type 1201, 1251, 2201 or 4201 processor is equipped with the Storage Protect Feature, certain operations are considered as "violations" of storage protection (e.g., the attempt to initiate a data transfer from a peripheral control to a starting location in the protected memory area). An internal interrupt signal is generated when such a violation occurs, and the contents of IIR and SR are automatically interchanged. The Resume Normal Mode instruction is used to return to the interrupted program.

A-ADDRESS REGISTER (AAR)

AAR normally contains the A-address portion of an instruction (i.e., the storage address of the rightmost character of the A-operand data). This address is loaded into AAR during the extraction phase of processing. In the execution of instructions whose operands are fields or rightmost-addressed fields or items, the contents of AAR are decremented by one as each character in the A field is manipulated.[2] The contents of AAR are incremented by one as each character in a record or leftmost-addressed field or item is extracted.[3]

B-ADDRESS REGISTER (BAR)

Normally the B-address portion of an instruction is loaded into BAR during the extraction phase. During the execution of most instructions, the contents of BAR are decremented by one as each character in the B field is extracted.[2] If the B operand is a record or a leftmost-addressed item, the contents of BAR are incremented by one as each character is extracted.[3]

SUMMARY

The foregoing information can be summarized as four easily remembered rules:

1. An instruction is read from left to right. As each character in the instruction is read, the contents of the sequence register are incremented by one.
2. A field is read from right to left.[2] As each character in a field is read, the contents of the corresponding address register are decremented by one.
3. A record is read from left to right.[3] As each character in a record is read, the contents of the corresponding current location counter are incremented by one.

[1] A Load Control Registers instruction can be used to store the desired op code address (see page 8-60).

[2] A field can also be moved internally from left to right by means of the Extended Move (EXM) Instruction (see Section VIII). In this case, the address register is incremented.

[3] A record can also be moved internally from right to left by means of the Extended Move instruction. In this case, the address register is decremented.

4. An item can be read either from left to right or from right to left. As each character in an item is read, the contents of the corresponding address register are incremented by one if reading from left to right, or decremented by one if reading from right to left.

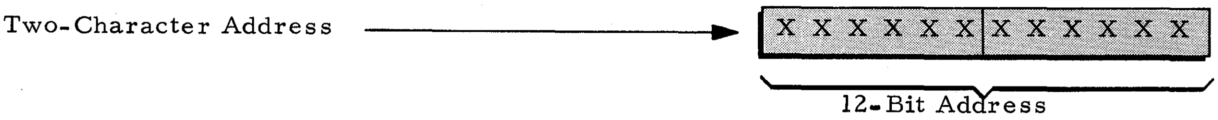
Recall that in all processors except the 4201, a control memory register is only as large as it need be to contain the largest main memory address in a user's processor (see Table 2-2), so that the size of the user's control registers ranges from 12 to 19 bits in length (control registers in a 4201 processor are always 19 bits long). The programmer should keep this fact in mind while reading the following description of addressing modes.

ADDRESSING MODES

As stated at the beginning of this section, an instruction is stored in a field of from 1 to 12 characters, depending on the instruction's format and the programmed addressing mode. The op code is stored as a single six-bit character. Variant characters or I/O control characters, if any, are each stored as single characters. The number of character locations in which each address portion is stored depends on the addressing mode selected by the programmer. This selection is made by means of a Change Addressing Mode instruction (see page 8-62), with which the programmer specifies the two-, three-, or four-character addressing mode. A significant feature of the Series 200 addressing technique is that the entire memory is directly addressable.

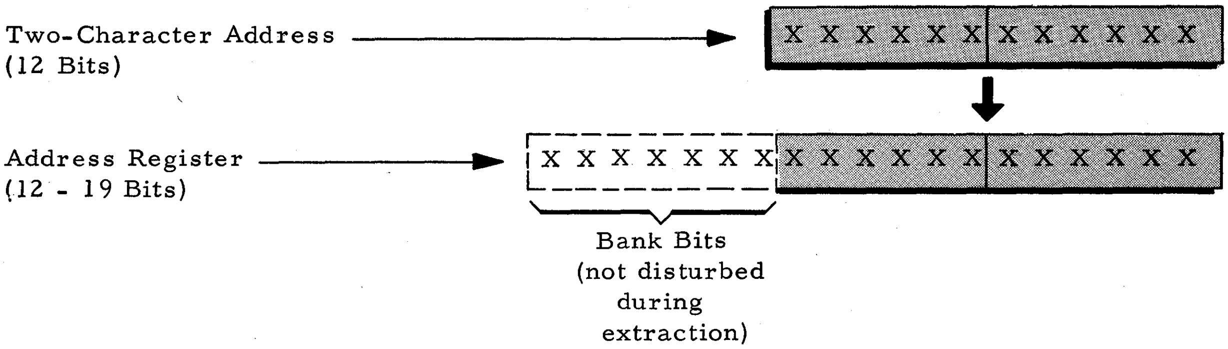
TWO-CHARACTER ADDRESSING MODE

An operand address written in the two-character addressing mode is stored in two consecutive character locations in memory. The stored address (a continuous 12-bit binary number) represents the address of a main memory location in the range 0 - 4,095_{10}.



During the extraction phase of instruction processing, the two-character address is placed in the rightmost 12 bit positions of the address register (AAR or BAR). Any bits in the register to the left of the two-character address are called "bank bits." Previous values in the bank bit positions of the register are not disturbed during instruction extraction.[1]

[1] The entire contents of an address register (bank bits + two-character address bits) are affected during the extraction of an instruction whose extraction path "duplicates A" (described on page 4-17). Extraction of all other two-character addresses affects only the rightmost 12 bits.



When the instruction is executed, the entire contents of the address register are interpreted as the operand address. Previous values in the bank bit positions, not disturbed during the extraction phase, are used to form the address of the operand during the execution phase. Thus, the bank bit values imply a base address to which the 12-bit address is added to form the actual operand address. If the bank bit values are all zeros, the 12-bit address is the actual operand address.

For example, a two-character A address specifying location $4,000_{10}$ is extracted and placed in AAR. The second bank bit in AAR (bit position 14) contains a residual value of "1", representing a base address of $8,192_{10}$. When the instruction is executed, the entire contents of AAR ($8,192_{10} + 4,000_{10}$) specify the address of the A operand — location $12,192_{10}$.

As the contents of the address register are incremented or decremented during "internal" execution, bank bits are not disturbed.[1] If the 12-bit address in the rightmost positions of the register becomes zero, a borrow from the first bank bit does not occur. Thus, the portion of memory which is addressable by a two-character address is the 4,096-character "bank" specified by the base address.

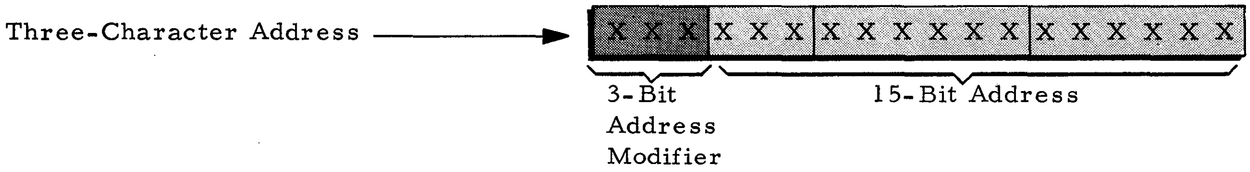
Indexed and indirect addressing (see below) cannot be performed in the two-character addressing mode.

THREE-CHARACTER ADDRESSING MODE

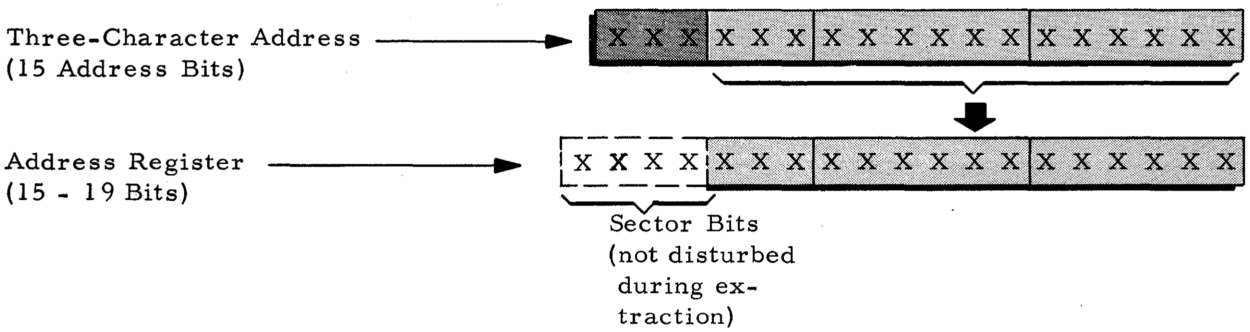
An operand address written in the three-character addressing mode is stored in three consecutive character locations of the memory. The rightmost 15 bits of the stored address represent the address of a main memory location in the range 0 - $32,767_{10}$. The leftmost three

[1] "Internal execution" is defined as the incrementing or decrementing of address register contents during memory cycles allocated to the central processor. When peripheral transfer operations are performed, using memory cycles allocated to read/write channels, incrementing and decrementing of address register contents affect all bits of the register. Thus, addressing during peripheral transfer operations is continuous throughout the memory.

bits, referred to as the "address modifier," specify whether the address is direct, indirect, or indexed (see "Address Modification," page 4-8).



During the extraction phase, the 15-bit address is placed in the rightmost bit positions of the operand address register. Any bits in the register to the left of these bit positions are called "sector bits." Previous values in the sector bit positions of the register are not disturbed during instruction extraction.[1]


Three-Character Address
(15 Address Bits)
X X X X X X X X X X X X X X X X X X
Address Register
(15 - 19 Bits)
X X X X X X X X X X X X X X X X X X X
Sector Bits
(not disturbed during extraction)

When the instruction is executed, the entire contents of the address register are interpreted as the operand address. Previous values in the sector bit positions, not disturbed during the extraction phase, are used to form the address of the operand during the execution phase. Thus, the sector bit values imply a base address to which the 15-bit address is added to form the actual operand address. If the sector bit values are all zeros, the 15-bit address is the operand address.

For example, a three-character A address specifying location $12,000_{10}$ is extracted and placed in AAR. The first sector bit in AAR (bit position 16) contains the value "1", representing a base address of $32,768_{10}$. When the instruction is executed, the entire contents of AAR ($32,768_{10} + 12,000_{10}$) specify the address of the A operand — location $44,768_{10}$.

As the contents of the address registers are incremented or decremented during "internal" execution, sector bits are not disturbed. If the 15-bit address in the rightmost locations of the address register becomes zero, a borrow from the first sector bit does not occur. Thus, the

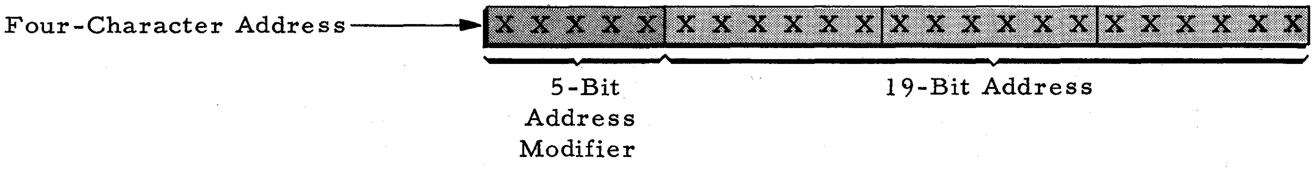
[1] The entire contents of an address register (sector bits + 15-bit address) are affected during the extraction of an instruction whose extraction path "duplicates A" (described on page 4-17). Extraction of all other three-character addresses affects only the rightmost 15 bits in the register.

largest portion of memory which is addressable by a three-character address is the 32,768-character "sector" specified by the base address.

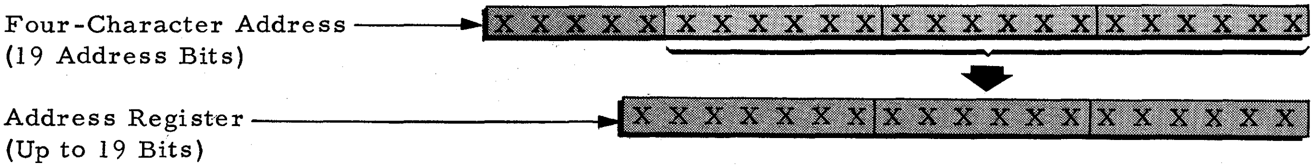
Addressing is continuous throughout the entire memory when a peripheral transfer operation is performed, as in the two-character mode.

FOUR-CHARACTER ADDRESSING MODE

An operand address written in the four-character addressing mode is stored in four consecutive character locations. The rightmost 19 bits represent a main memory address in the range 0 - 524,288$_{10}$. The leftmost five bits — the "address modifier" — specify whether the address is direct, indirect, or indexed (see "Address Modification," below).



The 19-bit address is placed in the address register during the extraction phase. Thus, the entire contents of the address register are affected during the extraction of a four-character address.


Four-Character Address (19 Address Bits)
X X X X X X X X X X X X X X X X X X X X X X X X
Address Register (Up to 19 Bits)
X X X X X X X X X X X X X X X X X X X

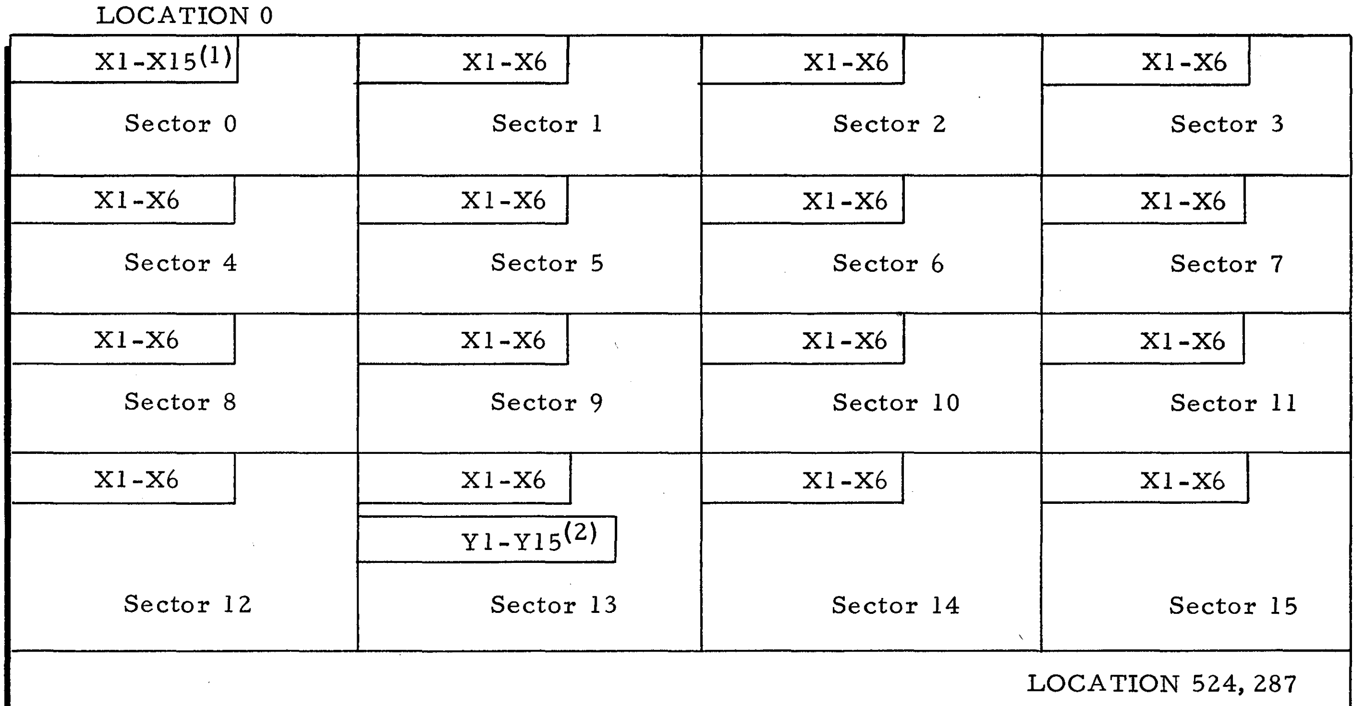
The entire contents of the register are interpreted as the operand address when the instruction is executed. As the contents of the operand address registers (AAR and BAR) are incremented or decremented during execution, all bits in the register are affected. Thus, addressing is continuous throughout the entire range of available memory (up to 524,288 locations) in the four-character addressing mode.

ADDRESS MODIFICATION

Indirect and indexed addressing can be used to modify three- or four-character addresses in any Model 200 processor containing the Advanced Programming Instructions (Feature 010 or 011) and in all Type 1201, 1251, 2201, and 4201 processors. These addressing forms are represented by the configuration of the "address modifier" as described below and are interpreted by th processor during the extraction phase.

INDEX REGISTERS

Index registers are used to store values to be used for address modification during instruction execution. A Series 200 processor can contain up to 120 index registers, depending on the type of processor and the optional features included in that processor. Figure 4-3 shows the memory areas utilized by the largest possible complement of index registers in a Series 200 memory. The portion of a processor's index register complement usable by a program at any given time varies with the program's location in main memory and the addressing mode in use. Table 4-1 summarizes the number of index registers simultaneously available to a program.



[1] Registers X1-X15 are not available in: (1) the Type 201 processor; and (2) the Type 201-1 or -2 processor not equipped with the Advanced Programming Instructions. In each of these cases, a group of index registers X1-X6 is located in place of X1-X15.

[2] Registers Y1-Y15 can be positioned, under program control, in the first 61 locations of any 4, 096-character bank of memory. If these registers are positioned in the first bank of a 32, 768-character sector, they replace the group of six index registers in that sector.

Figure 4-3. Series 200 Index Register Map

Index Register Map (Figure 4-3)

Registers X1-X6 are available to instructions executed in the three-character mode. These registers are located in the first 25 positions (locations 0 through 24) of the 32, 768-character sector in which the instruction is stored.[1] Since there can be as many as sixteen 32,768-character sectors in a Series 200 main memory, up to 96 index registers are supplied for use in three-character addressing mode.

Table 4-1. Number of Index Registers Simultaneously Available to a Program

Type	Basic Processor	Features		Minimum	Maximum
		Advanced Programming (Feature 010 or 011)	Storage Protect (Feature 1114, 1117, or 1118)		
201	0	6	n/a	0	6
201-1	0	15	n/a	0	15
201-2	0	15	n/a	0	15
1201	15	n/a[1]	15	15	30
1251	15	n/a[1]	15	15	30
2201	15	n/a[1]	15	15	30
4201	15	n/a[1]	15	15	30

[1] Advanced Programming is a standard feature on the Type 1201, 1251, 2201, and 4201 processors.

Index registers X1-X15, located in the first 61 character positions of memory, are available to instructions executed in the four-character addressing mode. The placement of these registers is independent of the location of the instruction whose address(es) is indexed. Registers Y1-Y15, located in the first 61 positions of a "protected" memory area, are available to all programs operating in the four-character addressing mode in processors equipped with the Storage Protect Feature.[2] The specific bank at which the protected memory area begins is specified by use of the Load Index/Barricade Register instruction (see Section VIII).

THREE-CHARACTER ADDRESS

The address modifier of a three-character address (i.e., the leftmost three bits of the stored address) specifies whether the address is direct (000), indirect (111), or indexed (001 through 110).

Indirect Addressing

In previous examples and illustrations in this section, an address portion of an instruction always specifies the address of a data field in the main memory. This manner of addressing an

[1] These registers are always located in the first 25 locations (locations 0-24) of memory in a Type 201 or 201-1 processor.

[2] Programs operating in the unprotected portion of memory can read the contents of Y1-Y15 but cannot write into these registers.

operand is commonly referred to as direct, or "first-level," addressing. In some instances, instead of specifying the location of a data field directly, it is more useful to be able to specify the storage location of another address, which in turn specifies the location of the desired data field. This manner of locating an operand is referred to as indirect, or "second-level," addressing.

A three-character indirect address is specified by an address modifier of all one bits and refers to the leftmost storage location of another main memory address. The referenced address can itself be direct, indirect, or indexed as specified by its address modifier. Thus, an indirect address can specify another indirect address, and so on through any number of levels, or it can specify an indexed address. The method of coding an indirect address is illustrated in Section 5.

Figure 4-4 shows the extraction of an Add instruction in which indirect addressing is specified in the A address and direct addressing is specified in the B address. Note that the A address (indirect) references the leftmost location of another main memory address. This address, in turn, specifies the location of the rightmost character in the A field. Note further that if the address modifier of location 1027 were not "000", the remainder of the stored address would be interpreted as an indexed or indirect address.

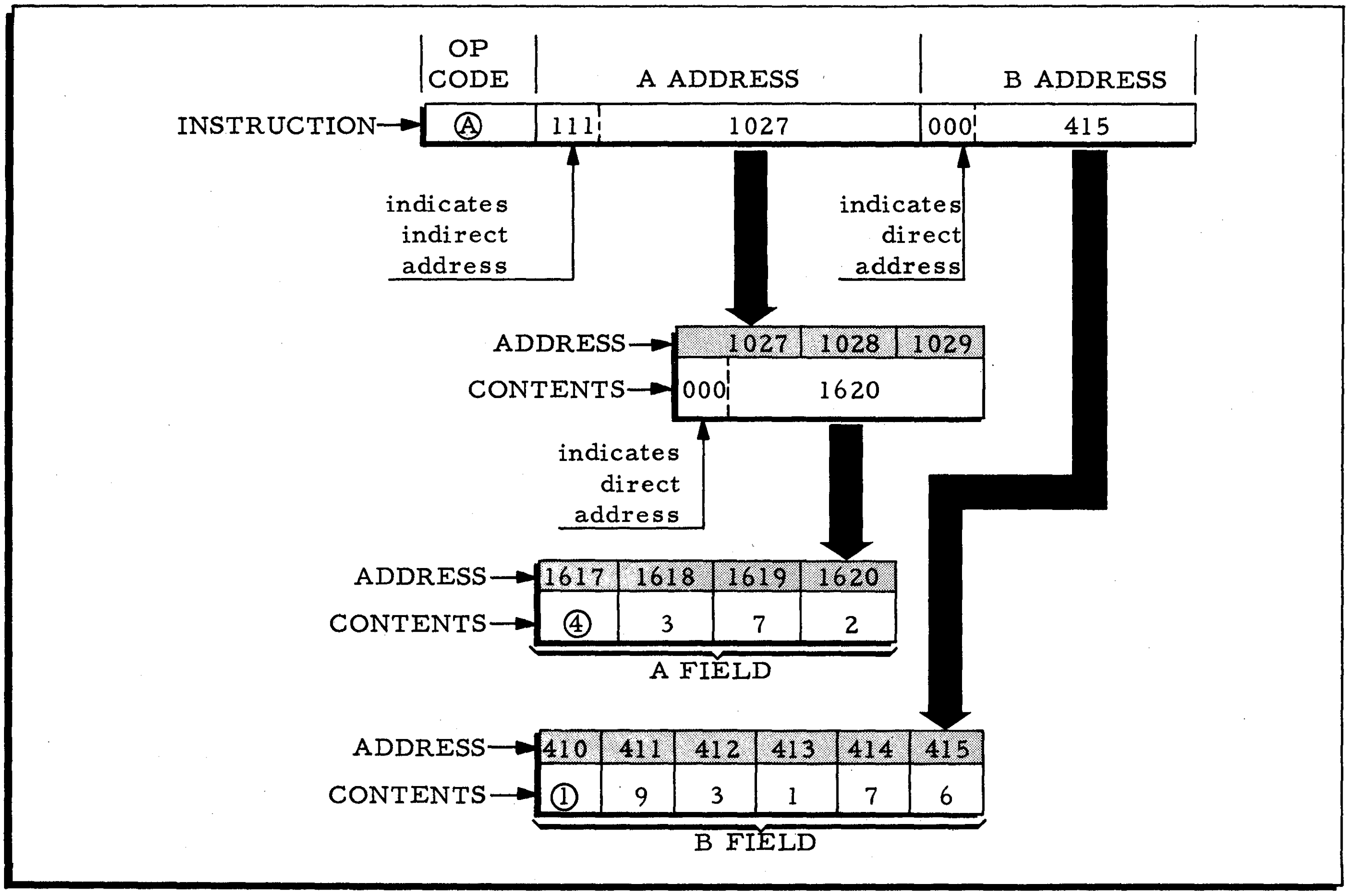


Figure 4-4. Extraction of Three-Character Indirect Address

Indexed Addressing

When indexed addressing is performed in the three-character mode, the rightmost 15-bit contents of an index register are automatically added to the 15-bit address field in an instruction. Three variables must be defined in any indexing operation: (1) the index register to be used, (2) the address to be modified, and (3) the factor (referred to as an augment) to be added to the address. The index register to be used is specified in the address modifier of an address field (see Table 4-2). The address to be modified can be stored in the same address field or it can be stored in the designated index register. If the address to be modified is stored in an address field, the augment is stored in the designated index register and vice versa.

Table 4-2. Index Register Addresses in Three-Character Addressing Mode

Index Register	Address Modifier	Storage Field	Address
X1	001	2 - 4 (+n)	4 (+n)
X2	010	6 - 8 (+n)	8 (+n)
X3	011	10 - 12 (+n)	12 (+n)
X4	100	14 - 16 (+n)	16 (+n)
X5	101	18 - 20 (+n)	20 (+n)
X6	110	22 - 24 (+n)	24 (+n)
n = first location of the 32,768-character sector in which the instruction is stored.			

The modification of an address occurs in its respective address register. For instance if the B-address portion of an instruction is indexed, the modification is performed in BAR. This means that neither the original instruction stored in the main memory nor the contents of the index register is altered in any way.

Normal programming, such as a load or a move operation, can be used to store a value in an index register. Similarly, the contents of an index register can be changed by using an instruction such as Binary Add or Binary Subtract. Note that since the index registers are located in the main memory, they can be used as normal storage locations when they are not being used for indexing operations.

Figure 4-5 illustrates how the Add instruction on page 4-11 would be extracted if indexed addressing were specified in the A-address portion of the instruction. The method of coding an indexed address is illustrated in Section V.

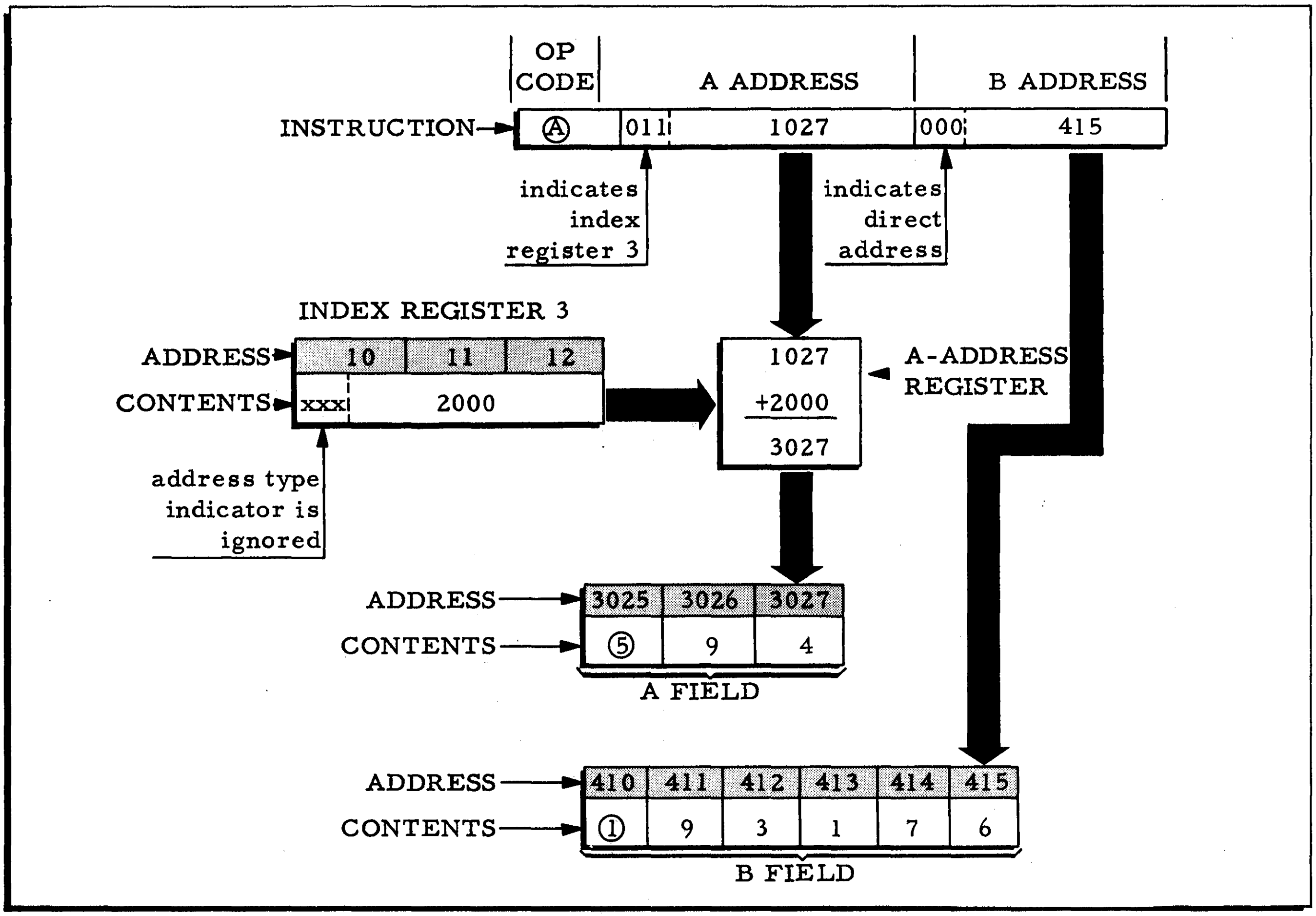


Figure 4-5. Extraction of Indexed Address in Three-Character Mode

FOUR-CHARACTER ADDRESSING MODE

The address modifier in a four-character address consists of the leftmost five bits of the address (see page 4-8). The configuration of these bits specifies whether the address is direct (00000), indirect (10000), or indexed (00001 through 11111, skipping over 10000).

Indirect Addressing

Indirect addressing in the four-character addressing mode is performed similarly to that in the three-character mode, except that:

1. a five-bit address modifier whose configuration is 10000 specifies indirect addressing; and
2. a four-character address is extracted.

The method of coding a four-character indirect address in Easycoder assembly language is identical to that used for a three-character indirect address (see Section V).

Indexed Addressing

Four-character indexed addresses to be modified by index registers X1 through X15 are specified by an address modifier whose configuration is 00001 through 01111, respectively. Index registers Y1 through Y15, when present, are specified by the configurations 10001 through 11111 (see Table 4-3). Register locations are shown in Figure 4-3.

Table 4-3. Index Register Addresses in Four-Character Addressing Mode

Index Register	Address Modifier	Storage Field	Address
X1	00001	1-4	4
X2	00010	5-8	8
X3	00011	9-12	12
X4	00100	13-16	16
X5	00101	17-20	20
X6	00110	21-24	24
X7	00111	25-28	28
X8	01000	29-32	32
X9	01001	33-36	36
X10	01010	37-40	40
X11	01011	41-44	44
X12	01100	45-48	48
X13	01101	49-52	52
X14	01110	53-56	56
X15	01111	57-60	60
Y1	10001	Same as above, only relative to the 4,096-character memory bank designated by the Load Index/Barricade Register instruction (see page 8-79).	
Y2	10010		
Y3	10011		
Y4	10100		
Y5	10101		
Y6	10110		
Y7	10111		
Y8	11000		
Y9	11001		
Y10	11010		
Y11	11011		
Y12	11100		
Y13	11101		
Y14	11110		
Y15	11111		

When indexed addressing is performed in the four-character mode, the contents of the specified index register are added to the address field of the instruction. However, only the number of active address bits of the index register and the address field are combined (i.e., only the number of bits which are required to address the entire memory of the user's processor). The number of active address bits corresponds to the size of a control memory register (see Table 4-4). In a 4201 processor, all 19 control register bits are active, regardless of main memory size.

Table 4-4. Active Address Bits in Series 200 Processors

Main Memory Capacity (Chars.)	32,768	65,536	131,072	262,144
Number of Active Address Bits	15	16	17	18

If the main memory capacity of a user's system lies somewhere between any two figures in the top row of Table 4-4, the larger number of active address bits is used. For instance, if a processor contains 49,152 characters, there are 16 active address bits in an index register (and in a control register).

The extraction of a Subtract instruction written in the four-character addressing mode is shown in Figure 4-6. Indirect addressing is specified in the A address, and indexed addressing (via index register X13) is specified in the B address.

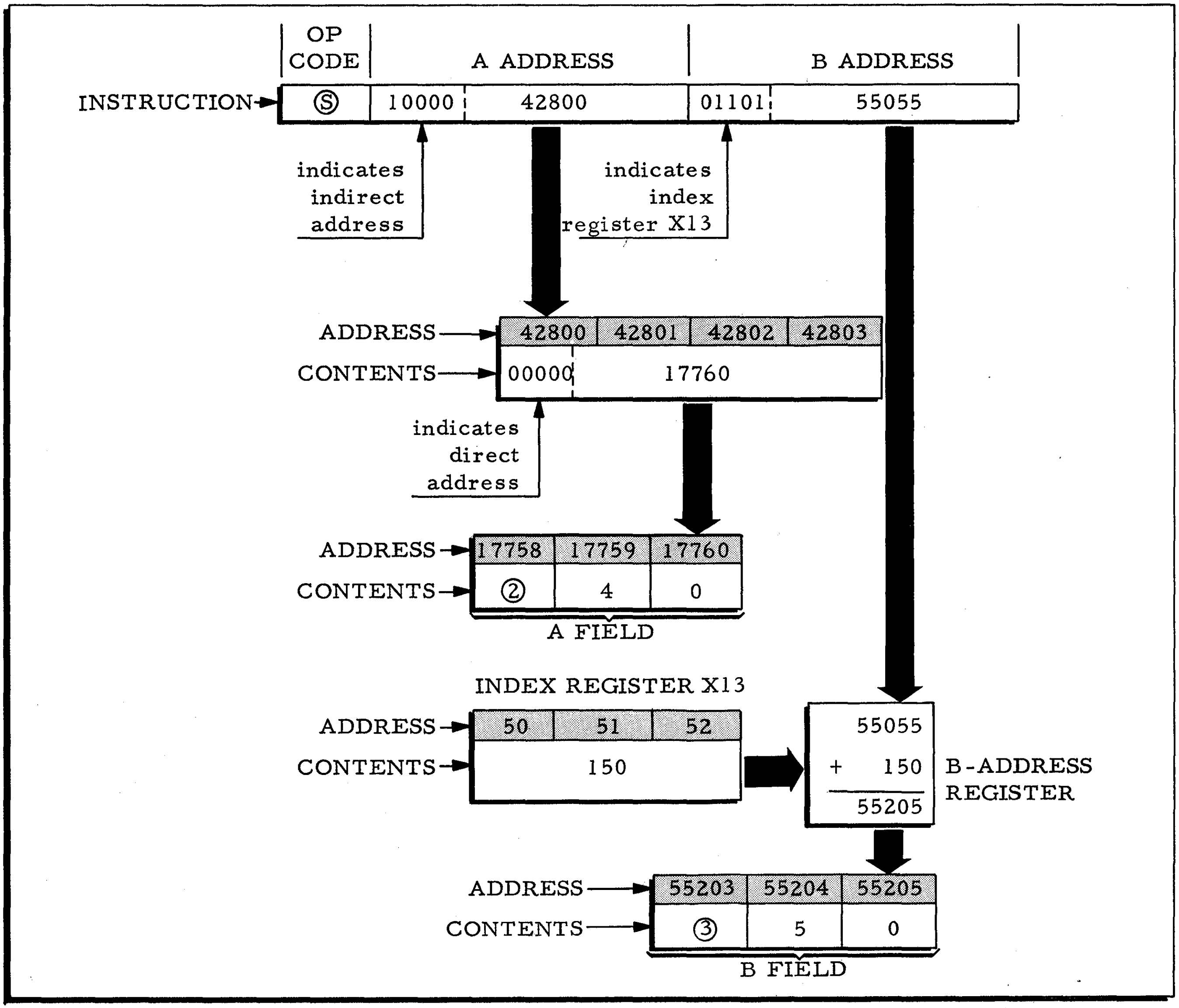


Figure 4-6. Extraction of Indirect and Indexed Four-Character Addresses

TREATMENT OF ADDRESSES LARGER THAN A MEMORY'S MAXIMUM ADDRESS

It is possible in some processors to specify in instructions direct addresses which are larger than the address of the processor's highest memory location. This condition can exist in all 4201 processors smaller than the maximum configuration and in any other Series 200 processor whose memory capacity is not a power of two.

Likewise, it is possible in any Series 200 processor, by the use of indexed addressing, to specify addresses and address modifiers whose sums are potentially greater than the address of the memory's highest location. For example, consider the case where, in a machine having a 49,152-character memory, an instruction contains the address 49,000 and the address is indexed using a register which contains the value 1,000. Obviously, the sum of 49,000 and 1,000 is greater than the memory's largest address, 49,151.

Situations such as the ones just cited are handled differently, depending upon the relationship between the potential address and the memory size involved and whether or not the Storage Protect Feature is installed and in effect. In particular, such situations can be categorized according to whether the potential address is larger or smaller than the range of addresses representable by active address-register bits.

Potential Addresses Within Address Register Range

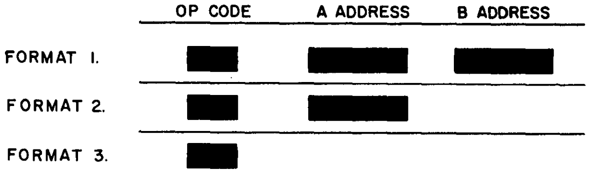
In a 4201 processor without storage protection in effect, encountering a simple, direct address, or the potential sum of an indexed address and index register contents, which lies between the address of the highest actual memory location and the address registers' upper limit, causes the processor to stop. Results are unspecified for the other Series 200 processors. Any Series 200 processor with Storage Protect in effect, upon encountering an address of the type described above, will perform the following actions: the II address violation indicator is set, the instruction is terminated prematurely, and an internal interrupt is generated (see Appendix E, page E-2).

Potential Addresses Outside Address Register Range

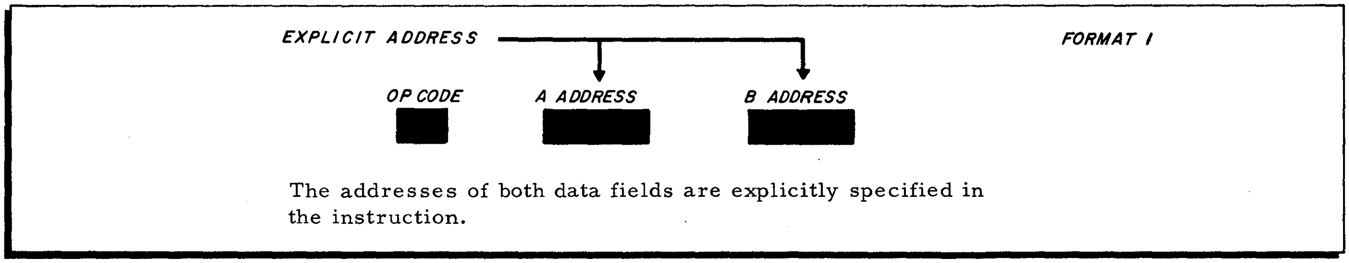
In any Series 200 processor, if a simple direct address, or the potential sum of an indexed address and index register contents, is greater than the largest address representable by active address-register bits, the resultant address is formed modulo the number of locations addressable with the active address bits; i.e., a memory "wrap-around" occurs. For example, in a 49K Model 2200, a total of 65,536 locations can be addressed by 16 active address bits. If, in such a machine, an address of 48,000 is indexed by the value 27,000, the resultant effective address will be 48,000 + 27,000 - 65,536, or 9464.

EXPLICIT ADDRESSING, IMPLICIT ADDRESSING, AND CHAINING

Consider the three instruction formats illustrated below.



Format 1 corresponds to the instructions used in the preceding illustrations. The significant feature of this format is that the addresses of both the A and the B data fields are explicitly specified in the instruction. For this reason the data fields are said to be "explicitly addressed." In general, whenever the programmer writes the address of a data field on his coding sheet, he is explicitly addressing that data field (see Figure 4-7).



The addresses of both data fields are explicitly specified in the instruction.

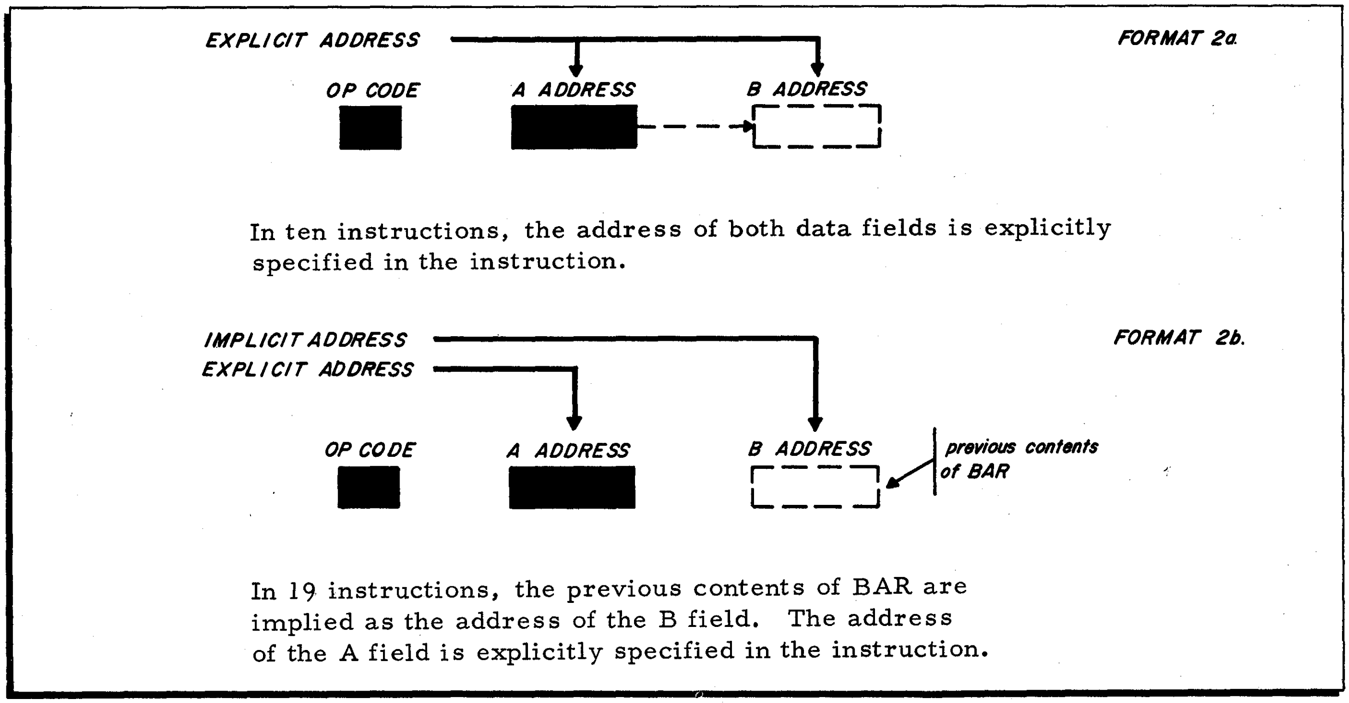
Figure 4-7. Series 200 Instruction Format 1

Format 2 has two possible interpretations (see Figure 4-8):

1. Ten Series 200 instructions coded in format 2 cause the A address to be loaded into both AAR and BAR.[1] Thus, although the B-address portion of the instruction is omitted, the B field is explicitly addressed by the A-address portion. The extraction path of these instructions is said to "duplicate A" (see Appendix C), since the contents of AAR are duplicated in BAR.

2. The A address of 19 instructions is loaded into AAR only, leaving BAR undisturbed. An omitted B address in any of these instructions implies that the previous contents of BAR will be used as the address of the B field. For this reason the B field is said to be "implicitly addressed," and the extraction path of these instructions "preserves B" (see Appendix C).

[1] The entire contents of AAR are loaded into BAR during extraction, so that all bit positions in BAR are identical to those in AAR.



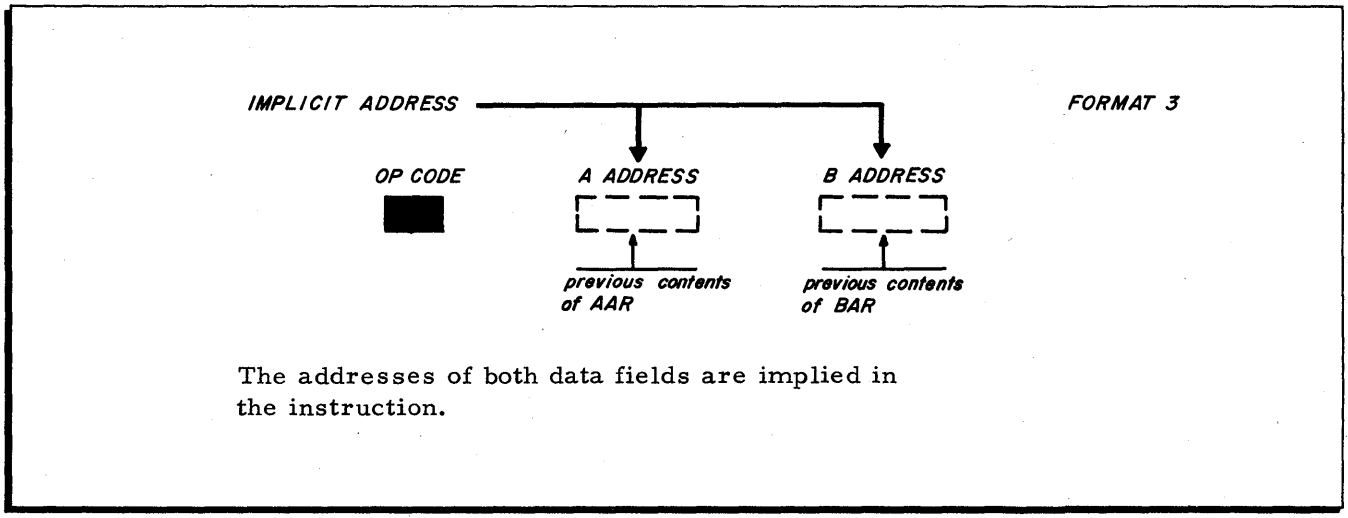
In ten instructions, the address of both data fields is explicitly specified in the instruction.

In 19 instructions, the previous contents of BAR are implied as the address of the B field. The address of the A field is explicitly specified in the instruction.

Figure 4-8. Series 200 Instruction Format 2

In format 3, both data fields are implicitly addressed. The previous contents of AAR are used as the address of the A field, and the previous contents of BAR are used as the address of the B field (see Figure 4-9).

Implicit addressing is extremely useful in situations where it is desired to perform a series of operations on data fields that are in consecutive storage locations. The use of implicit addressing reduces both the time required to perform the operations and the number of memory locations required to store the instructions.



The addresses of both data fields are implied in the instruction.

Figure 4-9. Series 200 Instruction Format 3

As an example, assume that three 10-character fields stored in sequence are to be added to three other sequential fields. First, examine how this operation would be performed using explicit addressing. Upon completion of the first instruction, AAR contains 890 and BAR contains 690. These are the same values that appear in the A- and B-address portions of the second instruction. Similarly, upon completion of the second instruction, AAR and BAR contain 880 and 680 — the A and B addresses of the third instruction. Since in each case AAR and BAR contain the addresses used in the next instruction, it is unnecessary to write these addresses in the instruction. In other words, this operation could be performed using implicit addressing in the second and third instructions.

```
Ⓐ    900    700
Ⓐ    890    690
Ⓐ    880    680
```

```
Ⓐ    900    700
Ⓐ
Ⓐ
```

Connecting instructions together so that the contents of AAR, BAR, and the variant register (see below) at the conclusion of one instruction satisfy the requirements of the next instruction is called "chaining." Using explicit addressing in the three-character addressing mode, 21 storage locations are required to store the instructions above and the operation takes 117 microseconds to complete on a Type 2201 processor. If the instructions were "chained," nine storage locations would be used and 105 microseconds would be required to complete the operation.

Instructions which require a variant character can also be chained by using the previous contents of the variant register. (The variant register is a single-character, internal register into which the variant character of an instruction is loaded during extraction.) The extent of chaining variant characters (i.e., the number of acceptable instruction formats in which the previous contents of the variant register can be used) varies with the processor being used.

In the Type 201-2, 1201, 1251, 2201, and 4201 processors, variant characters can be chained by an instruction coded in any format (i.e., format 1, 2, or 3 shown on page 4-17). The previous contents of the variant register are not normally distrubed by the processing of an instruction which does not contain a variant character (see the instruction Branch, Move Characters and Edit, and Move and Translate for exceptions).

In the Type 201 and 201-1 processors, the previous contents of the variant register are destroyed by the processing of an instruction which contains an address portion. Thus, the only instructions which can chain variant characters in these processors are those instructions coded without address portions (i.e., format 3 on page 4-17).

Chaining is not limited to sequential operations having the same op code. The only condition that must be met is that an instruction must leave the contents of AAR, BAR, and, if required, the variant register such that they satisfy the addressing requirements of the next instruction in sequence.

To enable the programmer to chain instructions wherever possible, the description of each instruction (see Section VIII) includes a table showing the contents of the address registers after the instruction has been executed. Also, Appendix C denotes whether each instruction in the machine complement can or cannot be chained.

5 EASYCODER PROGRAMMING

INTRODUCTION

The preparation of Series 200 programs is greatly simplified by the use of Easycoder — a concise, easy-to-use programming system. Specifically, Easycoder relieves the programmer of many time-consuming duties associated with writing a program in actual machine language. It makes it unnecessary, for example, to maintain a careful record of the storage address assigned to each instruction. In addition, it allows the programmer to employ meaningful symbolic tags (e.g., TAX, FICA, and TOTAL) to specify data, rather than using absolute memory addresses. In situations where a stored program must be relocated or modified, Easycoder can be used to perform the required alterations automatically.

Easycoder includes a number of assembly systems; these systems are:

- EASYCODER A: Part of the SERIES 200/BASIC PROGRAMMING SYSTEM. Easycoder A operates in a system having a minimum main memory size of 4,096 characters. (Additional memory, however, may be used to advantage.) For additional information refer to Easycoder A Assembly System (Order No. 490).

  NOTE: A counterpart of Easycoder A — Easycoder A (P) — is available for use in a paper tape environment. The main memory requirements are identical to those of Easycoder A. See Easycoder A(P) Assembly System (Order No. 695) for more information.

- EASYCODER B: Also part of the SERIES 200/BASIC PROGRAMMING SYSTEM. Easycoder B operates in a system having a minimum main memory size of 8,192 characters. (Additional memory may be used to advantage, however.) See Easycoder B Assembly System (Order No. 011) for additional information.

- EASYCODER C: Part of the SERIES 200/OPERATING SYSTEM - MOD 1. Easycoder C operates in a system having a minimum of 12,288 characters of main memory. (Additional memory, however, may be used to advantage.) For additional information refer to Easycoder Assemblers C and D (Order No. 041).

- EASYCODER D: Part of the SERIES 200/OPERATING SYSTEM - MOD 1. Easycoder D operates in a system having a minimum of 16,384 characters of main memory. (Additional memory, however, may be used to advantage.) For additional information see Easycoder Assemblers C and D (Order No. 041).

Each assembly system includes two basic elements: the Easycoder symbolic language and an Easycoder Assembler. The Easycoder language is used to write the symbolic program (the source program) while the assembler translates the source program into the actual machine-language program (the object program).

To prepare a program in Easycoder symbolic language, the programmer uses an Easycoder Coding Form (see Figure 5-5) and enters each symbolic instruction or definition on a separate line. As a general rule, the instructions are written in the order in which they are to be executed. (However, the instructions must be in the proper sequence prior to assembly.) After the symbolic program has been written, each line of symbolic coding is punched into a separate source-program card. These cards are the input data which will be processed by an Easycoder assembler.

The assembler accepts the source-program cards and automatically produces a corresponding machine-language object program. It converts mnemonic operation codes into machine-language codes, assigns absolute storage addresses to instructions and symbolic operand references, and completely assembles the final program, storing it on punched cards or magnetic tape. Another output of the assembler may be a complete printed summary of the symbolic source program and the corresponding machine-language entries. Figure 5-1 illustrates the relationship of the source program, assembler and object program.

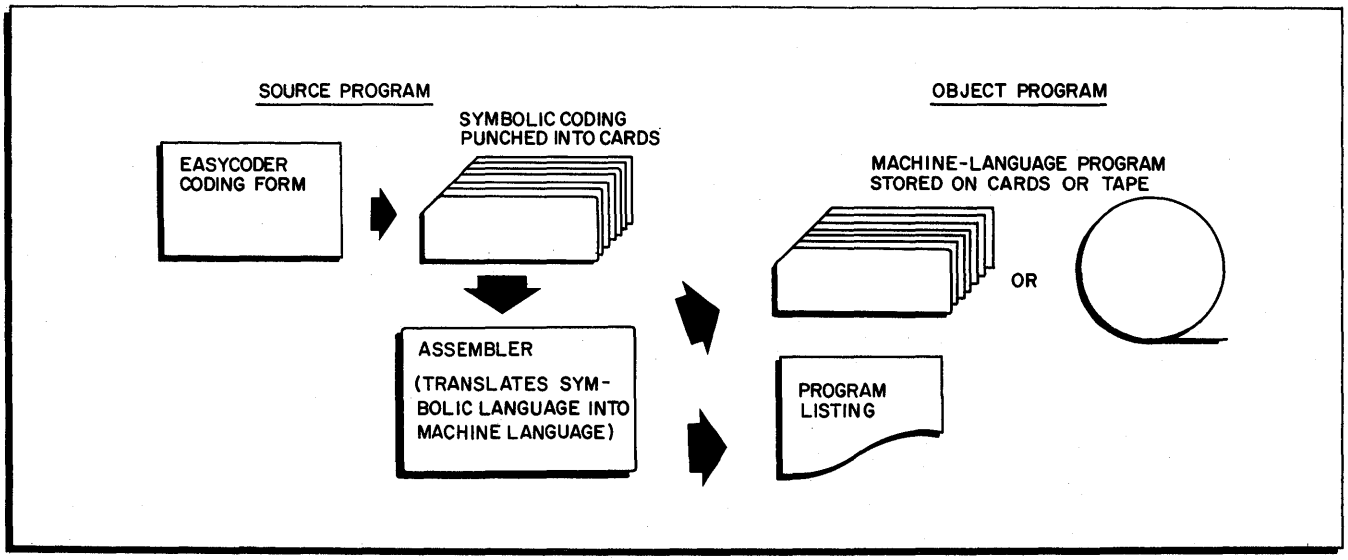

SOURCE PROGRAM
EASYCODER CODING FORM
SYMBOLIC CODING PUNCHED INTO CARDS
ASSEMBLER
(TRANSLATES SYMBOLIC LANGUAGE INTO MACHINE LANGUAGE)
OBJECT PROGRAM
MACHINE-LANGUAGE PROGRAM STORED ON CARDS OR TAPE
OR
PROGRAM LISTING

Figure 5-1. Relationship of Source Program, Assembler, and Object Program

THE SYMBOLIC LANGUAGE

The Easycoder symbolic language is composed of a set of mnemonic operation codes and a set of rules for defining memory areas, addressing operands, and entering constants. The mnemonic operation codes are predefined abbreviations for machine-language operation codes and, in general, provide an easily remembered description of each instruction. For example, SI is the Easycoder mnemonic for the Set Item Mark instruction, and BCC is the mnemonic for the Branch on Character Condition instruction. The set of rules includes special mnemonics for defining work areas in the main memory and for defining programmer-specified constants.

The statements used in writing an Easycoder program can be classified into three groups:

1. Data formatting statements make it possible to reserve areas and store constants without regard to their actual locations in memory. Data formatting statements are described in Section VI.
2. Assembly control statements are used by the programmer to control the assembly of his program. Assembly control statements are described in Section VII.
3. Data processing statements are the actual machine instructions to be executed in the object program. Section VIII contains a description of the data processing statements employed by the Models 200, 1200, 1250, 2200, and 4200.

THE ASSEMBLERS

The assembler element of each Easycoder assembly system translates the symbolic source program (written on the Easycoder Coding Form and subsequently punched into a source-program card deck) into machine-language entries, placing the resultant object program on either punched cards or magnetic tape. In addition to the object-program output, the assembler may also produce a printed listing containing the symbolic source program and the corresponding object-program entries (see Figures 5-2 and 5-3).

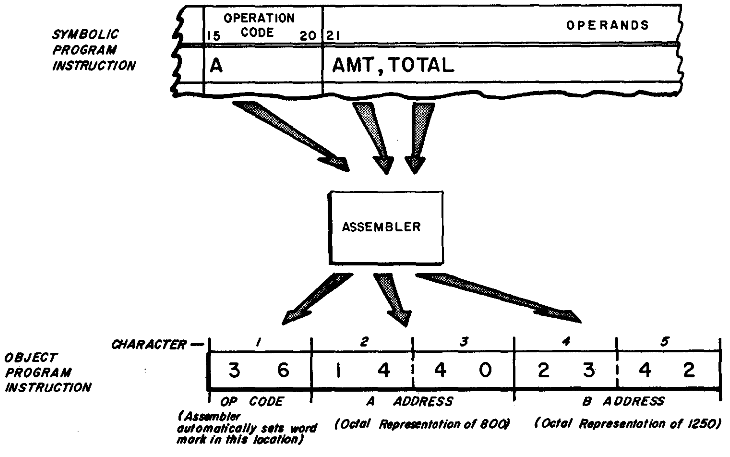


Figure 5-2. Two-Character Address Assembly

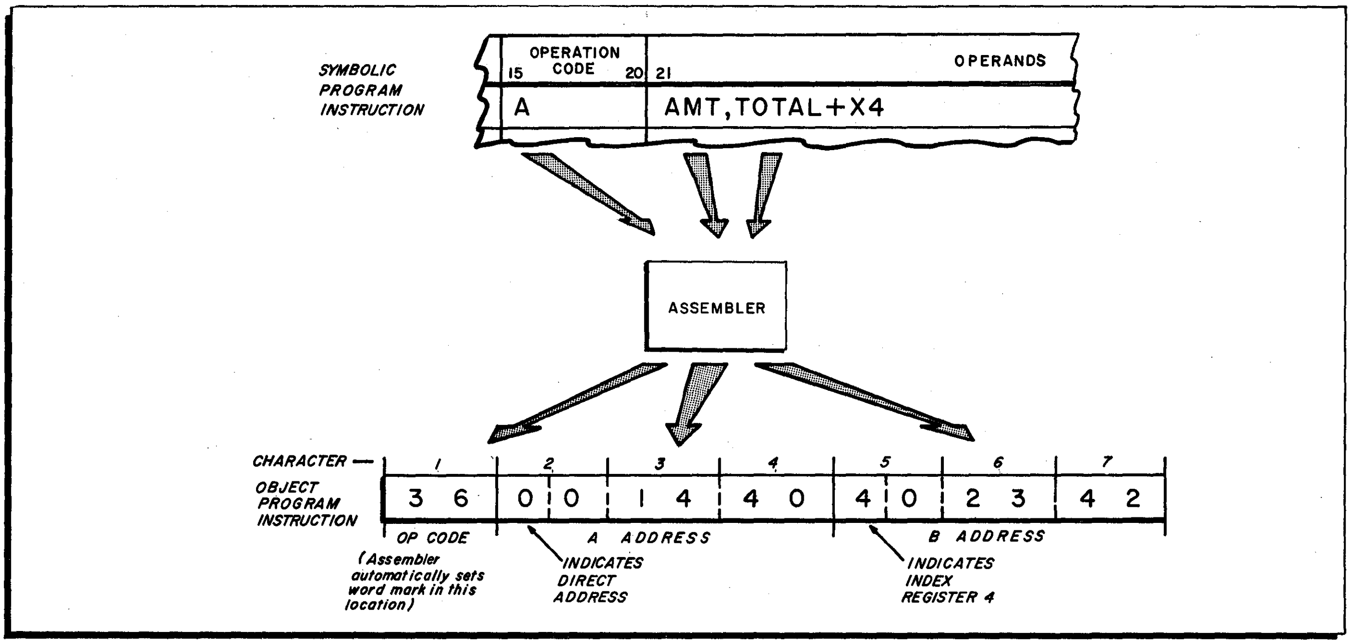


Figure 5-3. Three-Character Address Assembly

Figure 5-2 illustrates how an assembler assembles an object-program instruction using two-character address assembly. Assume that the tag AMT is assigned to memory location 800 and that the tag TOTAL is assigned to memory location 1250. Figure 5-3 shows how the assembler assembles an object-program instruction using three-character address assembly. Four-character addresses are assembled as shown in Figure 5-4. Assume that, in Figures 5-3 and 5-4, the tags are assigned the same values as in Figure 5-2.

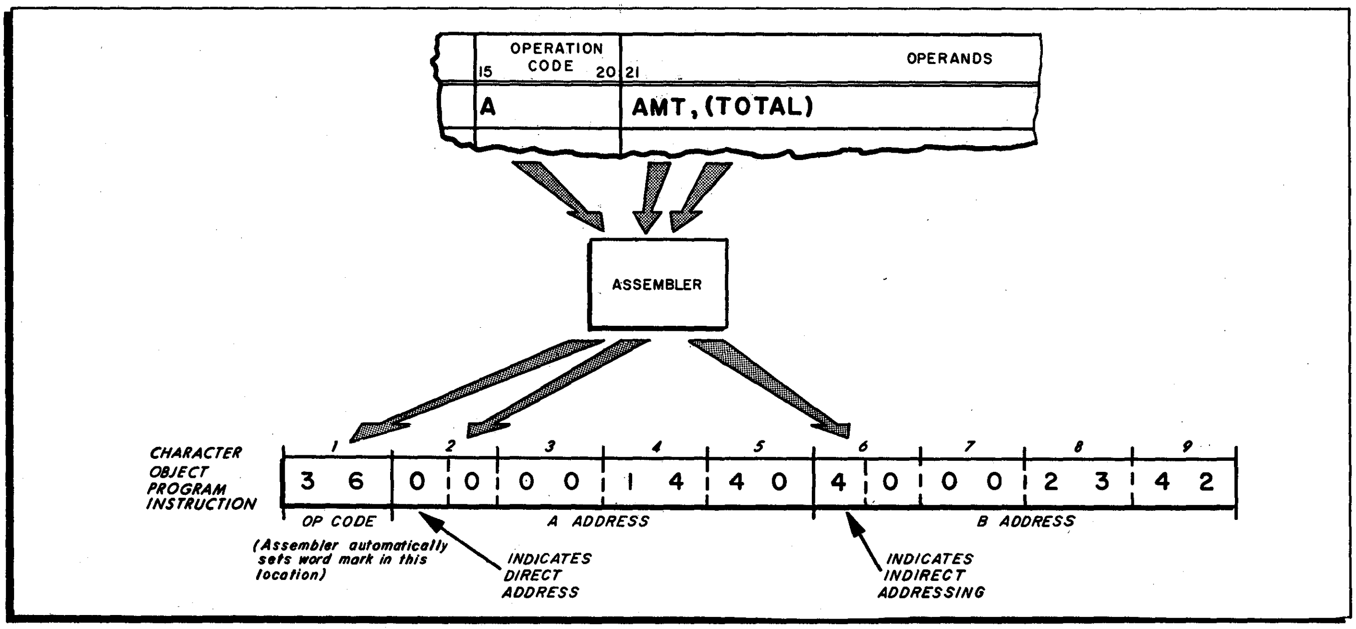
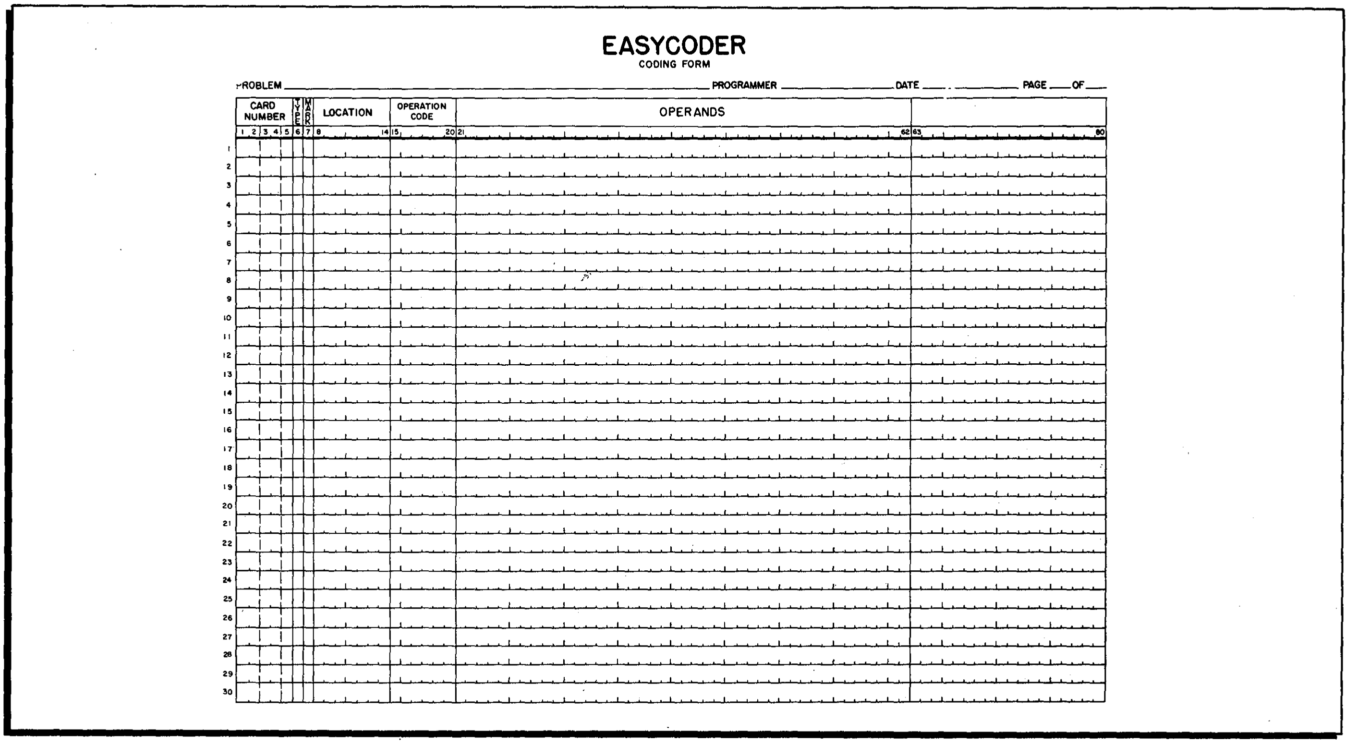


Figure 5-4. Four-Character Address Assembly

CODING FORM

Programs are written on the Easycoder Coding Form (Figure 5-5). This form is composed of fixed-format fields for coding such entries as card number, location, and operation code, and a variable-format field for operand addresses and comments. The numbers associated with each subdivision, or field, on the coding form indicate the card columns into which the characters written by the programmer are to be punched.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER 1 2 3 4 5	TYPE 6	MARK 7	LOCATION 8 14	OPERATION CODE 15 20	OPERANDS 21 62	63 80
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							
16							
17							
18							
19							
20							
21							
22							
23							
24							
25							
26							
27							
28							
29							
30							

Figure 5-5. Easycoder Coding Form

CARD NUMBER (Card Columns 1-5)

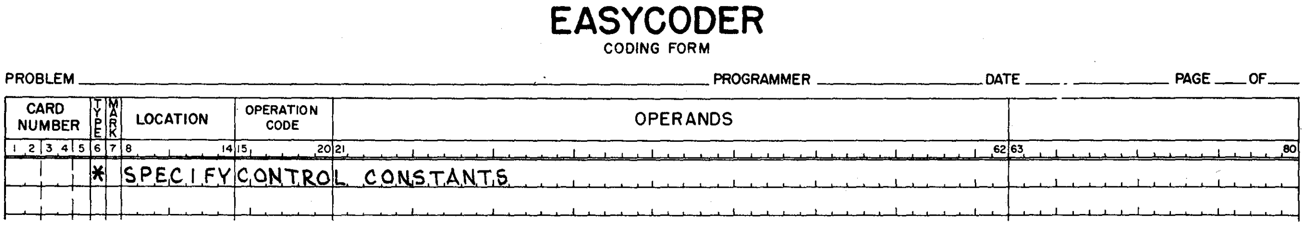
This five-character field is divided into three parts: the first two characters are used for page numbering, the next two for line numbering, and the last character for insertions. The page entry provides the proper sequencing of coding forms. The line number entry is used for the sequential numbering of instructions on each coding form. The single-character insertion entry permits one or more lines of coding to be inserted between existing lines. For example, to insert a line of coding between lines 16 and 17 of page 8, the following coding could be used.

CARD NUMBER PAGE 1 2	 LINE 3 4	 INS 5	TYPE
Ø 8	1 6		
Ø 8	1 6	5	
Ø 8	1 7		

NOTE: The number 5, which appears in column 5 above, is optional. An insertion may be made using any decimal, alphabetic, or special character. Provided that the characters are in ascending order of value (beginning with 0), multiple insertions may be made between any two instructions.

TYPE (Card Column 6)

For all instructions and constants, this column remains blank. However, the programmer can enter lines of descriptive information, called remarks lines, anywhere in the source program. Such a line, containing only descriptive data within columns 8 through 80, is identified by an asterisk (*) in column 6. Information inserted in this manner, while it remains as part of the source program, does not appear in the object program; it does, however, appear in the program listing.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1		*		SPECIFY	CONTRO	L CONSTANTS	
2							

Easycoder C and D Options

For Easycoder C or D users, this column may also contain the letter T to designate a temporary remarks card, or the letter D to designate a data card. If the programmer wishes to enter remarks lines anywhere in the source program but does not want these remarks to become a permanent part of the source program, a T instead of an asterisk (*) is placed in column 6. Remarks lines inserted in this manner are used only on the first assembly (i.e., when the program is being "inserted"), and are subsequently deleted from the symbolic program tape by the assembler. A temporary (like a permanent) remarks statement, while it appears in the program listing, does not appear in the object program.

A letter D in column 6 indicates a data card. All data cards must be contained in segments consisting only of data cards. In addition, any data card (or group of data cards) must be immediately preceded by a SEG card and immediately followed by either an EX, XFR, or END card. When a data card is encountered by an assembler, columns 8 through 80 are reproduced, unaltered, on the binary run tape or machine-language punched deck.

MARK (Card Column 7)

This field, used in conjunction with data formatting operations (described in Section VI), serves to set up required punctuation. Two sets of punctuation indicators are available; set I may be employed with all Easycoder assembly systems (A, B, C, and D); set II, however, may only be used with the Easycoder C and D. Both punctuation sets are described below.

Set I, consisting of a blank (Δ), an L, and an R, establishes the position of the item mark when defining an item (see Table 5-1). Word marking for this first set depends upon the class of instruction, as determined by the contents of the op code field.

NOTE: When an L is used and the leftmost (high-order) character is automatically word marked, a record mark will result.

Table 5-1. Set I Punctuation Indicators

Column 7 Contents	Resultant Item Mark Setting	
	Leftmost (High-order) Character	Rightmost (Low-order) Character
Δ	Δ	Δ
L	Item Mark	Δ
R	Δ	Item Mark

Set II, designed for use with the Easycoder Assemblers C and D, can be employed in situations which warrant unusual punctuation requirements. With this set (listed in Table 5-2), any one punctuation indicator controls the complete punctuation setting for the particular instruction or constant. However, there is no implicit word mark setting as in the first set. In other words, this second set of punctuation is not dependent upon the class of instructions.

Table 5-2. Set II Punctuation Indicators (Easycoder C and D Only)

Column 7 Contents	Resultant Punctuation Setting	
	Leftmost (High-order) Character	Rightmost (Low-order) Character
A	Word Mark	Δ
B	Item Mark	Δ
C	Record Mark	Δ
D	Δ	Word Mark
E	Δ	Item Mark
F	Δ	Record Mark
G	Item Mark	Item Mark
H	Item Mark	Word Mark
I	Item Mark	Record Mark
J	Word Mark	Item Mark
K	Word Mark	Word Mark
M	Word Mark	Record Mark
N	Δ	Δ
P	Record Mark	Word Mark
S	Record Mark	Item Mark
T	Record Mark	Record Mark

LOCATION (Card Columns 8-14)

The location field can contain an absolute memory address or a symbolic tag, or it can be left blank. An absolute memory address (expressed as a decimal number) specifies that the instruction or data will be stored in that location. No leading zeros are necessary when writing an absolute decimal number. Moreover, this type of entry does not affect the allocation of any subsequent instructions.

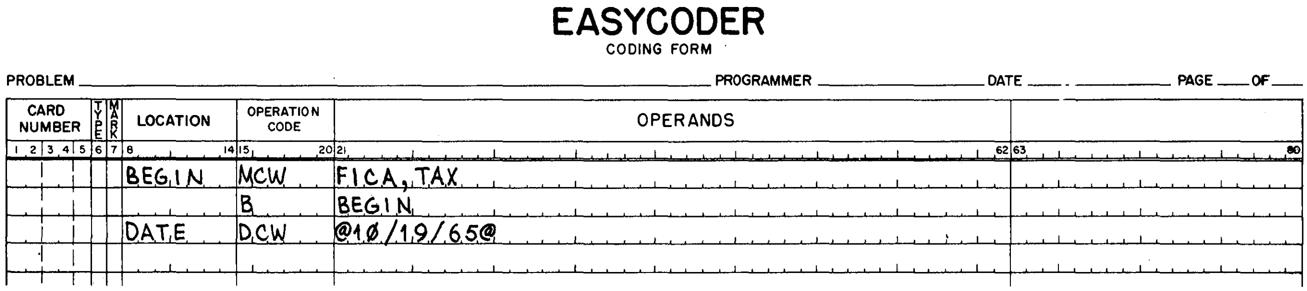
Symbolic tags provide simple, meaningful symbolic references for storage locations, constants, and instructions that are referred to elsewhere in the program. All symbolic tags written in the location field are assigned absolute addresses during assembly. When an entry is assigned a symbolic tag, the contents of the entry can then be referred to by that tag. This means that the programmer can refer to data via a symbolic tag and need not be concerned with its actual main memory address. One to six characters make up a symbolic tag (Easycoder D, however, can process tags of up to ten characters in length; see "Easycoder D Options" below). These characters can be alphabetic (A to Z) or numeric (0 to 9); the first character of the tag, however, must be alphabetic.

If the location field entry is made beginning in column 8, the following rules apply:

1. An absolute memory address assigned to an instruction refers to the leftmost character in the instruction.
2. An absolute memory address assigned to a constant or reserved area refers to the rightmost character in the field.
3. If a symbolic tag is assigned to an instruction, the address assigned to the tag will be the address of the leftmost character in the instruction.
4. If a symbolic tag is assigned to a constant or reserved area, the address assigned to the tag will be the rightmost character in the field.

These address assignment conventions can be reversed by leaving column 8 blank and entering the first character in column 9. In this case, the following rules apply:

1. An absolute memory address assigned to an instruction refers to the rightmost character in the instruction.
2. An absolute memory address assigned to a constant or reserved area refers to the leftmost character in the field.
3. If a symbolic tag is assigned to an instruction, the address assigned to the tag will be the address of the rightmost character in the instruction.
4. If a symbolic tag is assigned to a constant or reserved area, the address assigned to the tag will be the leftmost character in the field.



EASYCODER

CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER 1-5	TYPE 6	MARK 7	LOCATION 8-14	OPERATION CODE 15-20	OPERANDS 21-62	63-80
1				BEGIN	MCW	FICA,TAX	
2					B	BEGIN	
3				DATE	DCW	@1Ø/19/65@	
4							
5							

The first instruction shown above moves the contents of the field tagged FICA to the field tagged TAX. This instruction can be referred to in the operands field of another symbolic program entry via the tag BEGIN. For instance, the second instruction causes the program to branch to the MCW instruction by referring to it via its symbolic tag (BEGIN). In other words, the address of the operation code of the MCW instruction is inserted in an object-program instruction wherever the tag BEGIN appears as an operand in a symbolic-program entry. The third instruction defines an alphanumeric constant which can be referred to in the operand field of another symbolic-program entry via the tag DATE. In this case, the tag refers to the address of the rightmost character in the constant.

Easycoder C and D Options

Users of Easycoder C or D may also include, in the location field, an apostrophe (')[1] followed by a decimal number; this procedure serves to indicate an address relative to the out-of-sequence base (OSB). The out-of-sequence base, a value maintained by the assembler can be set by the XBASE instruction (see page 7-18). The assembler assigns to the corresponding statement an address equal to the sum of the decimal number and the current value of the OSB. (Leading zeros may be omitted from the decimal number.) The allocation of any subsequent instructions is not affected.

If the apostrophe and decimal number are written beginning in column 8, the following rules apply:

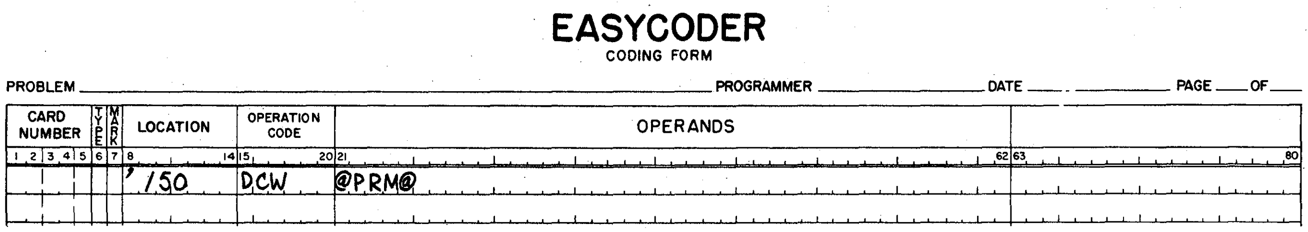
1. An address relative to the out-of-sequence base assigned to an instruction refers to the leftmost character in the instruction.
2. An address relative to the out-of-sequence base assigned to a constant or reserved area refers to the rightmost character of the field.

These address conventions can be reversed by leaving column 8 blank and entering the first character (the apostrophe) in column 9. In this case, the following rules apply:

[1] Card code 8, 2 (octal 12).

1. An address relative to the out-of-sequence base assigned to an instruction refers to the rightmost character in the instruction.
2. An address relative to the out-of-sequence base assigned to a constant or reserved area refers to the leftmost character of the field.

Assume, for example, that the OSB has been set to the value 500 by the last XBASE instruction. The following DCW statement is now encountered. The constant PRM is assigned, by the assembler, to locations 648 through 650. (The value of the OSB remains 500).



EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ____ OF ____

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				'150	DCW	@PRM@	
2							

Easycoder D Options

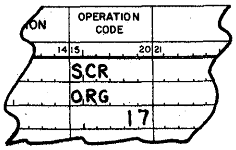
Symbolic tags of up to ten characters in length may be employed with Easycoder D. For symbolic tags consisting of six characters or less, the standard coding format is used. However, if tags of from seven to ten characters are used, the location field is modified such that it now occupies card columns 8-18. (This alternate format also requires that the operation code field and operands field be modified to accommodate the increase in tag size.) The same programming conventions which apply to six-character tags apply also to ten-character tags.

NOTES: 1. The program header (PROG) card is used to denote that the alternate format is to be employed. See page 7-2 for instructions on how to employ the PROG card in this manner.

2. Symbolic tags of more than six characters in length may not be used if the input is to be in the form of paper tape.

OPERATION CODE (Card Columns 15-20)

This six-character field can contain a mnemonic operation code for a machine instruction, an assembly program directive, or a data formatting code (see entries below). These entries must be left-justified. Machine-language operation codes (in octal notation) may be used instead of mnemonic codes. These octal codes are written in columns 19 and 20 of the operation code field, and columns 15 to 18 are left blank.



ON	OPERATION CODE	
14	15 20	21
	SCR	
	ORG	
	17	

Easycoder D Options

If the alternate coding format is used (i.e., the location field contains tags of from seven to ten characters in length), the operation code field occupies card columns 19-24. The method of coding mnemonic operation codes remains the same. If octal operation codes are used, they are written in columns 23 and 24; columns 19-22 are left blank.

OPERANDS

The operands field is a variable-format field which can contain a series of entries separate by commas and terminated by the first blank following any character other than a comma or a blank. In general, the operands field contains such entries as the addresses (either symbolic or absolute) of the data to be operated upon by a command in the operation code field, literals, address constants, or input/output information. Relative, indexed, and indirect addressing can be used in conjunction with absolute or symbolic addresses (see below).

Easycoder A and B (Operands Field: Card Column 21-62)

For either of these two assembly systems, column 62 terminates the operands field. Any punches appearing in columns 63-80 (of any line other than a remarks line) are ignored and do not even appear in the object-program listing. Remarks may be entered following the terminating blank.

Easycoder C and D (Operands Field: Card Columns 21-80)

For users of Easycoder C or D, the operands field extends to column 80. Remarks may be entered following the terminating blank. One or both operands can be bypassed during assembly by writing one or two leading commas, respectively, in the operands field. Such a comma, or commas, must be left-justified in the operands field and must be followed immediately (i.e., without intervening blanks) by any remaining entries, other than remarks.

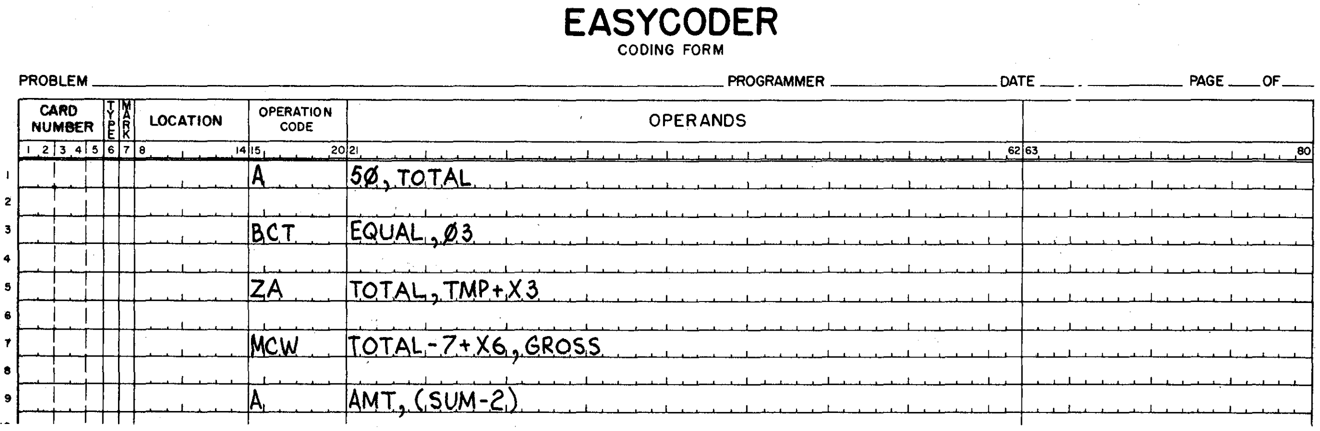
Easycoder D Options

If the alternate coding format is used (i.e., the location field contains tags of from seven to ten characters in length), the operands field occupies card columns 25-80. The method of coding entries and remarks remains the same.

Examples

The first sample instruction causes the contents of the field whose rightmost character is stored in memory location 50 to be added algebraically to the contents of the field designated by the tag TOTAL.

The second instruction tests the indicator specified by variant character 3 and branches to the address tagged EQUAL if the indicator is on.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					A	5Ø,TOTAL	
2							
3					BCT	EQUAL,Ø3	
4							
5					ZA	TOTAL,TMP+X3	
6							
7					MCW	TOTAL-7+X6,GROSS	
8							
9					A	AMT,(SUM-2)	

The third line of coding above shows an instruction in which the B address is indexed. The instruction causes the contents of field tagged TOTAL to be placed in the field designated by the tag TMP as modified by the contents of index register X3.

The fourth line of coding shows relative addressing and indexing being performed on the A address. The instruction causes the address seven before that tagged TOTAL to be modified by the contents of index register X6. The resultant address specifies a field whose contents are then placed in the field tagged GROSS. Assuming that TOTAL corresponds to memory location 540 and index register X6 contains a value of 80, the resultant address of this instruction would be 613.

The last line of coding above illustrates an instruction with indirect addressing on the B address. The contents of the field tagged AMT are added algebraically to the contents of the field whose address is stored in the field tagged SUM-2.

ADDITIONAL CODING RULES

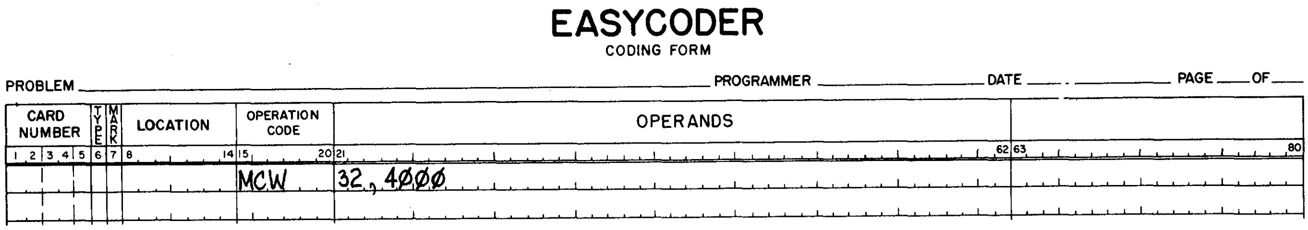
1. Comments and remarks can appear on any line following the last entry on that line and separated from it by a blank space. These notes will be printed on the program listing but will not be assembled as object-program entries. As mentioned previously, any line of coding containing only comments must be designated by an asterisk (*) or the letter T in column 6.
2. Any number of blank spaces may be used between the comma which terminates the A operand and the first character of the B operand. Similarly, any number of spaces may be used between the comma that terminates the B operand and a variant character.

ADDRESS CODES

Several types of address codes are valid in the operands field of an Easycoder statement. These codes are defined and illustrated below.

ABSOLUTE

The actual address of a character position in the main memory can be represented as a decimal number; leading zeros can be omitted. The sample instruction causes the contents of the field whose rightmost character location is 32 to be moved to the field whose rightmost character location is 4000.



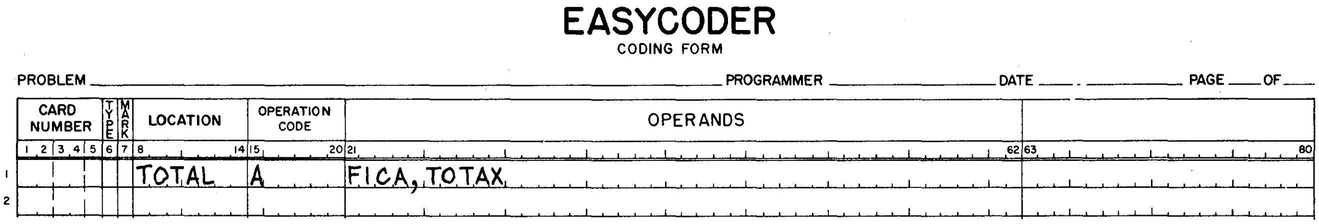
EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					MCW	32, 4ØØØ	
2							

SYMBOLIC

A symbolic address, or tag, can be used in the operands field only if it appears in the location field elsewhere in the symbolic program. In effect, a tag must be defined (by writing it in the location field of a symbolic entry) in order for it to be used as an operand address.



EASYCODER
CODING FORM

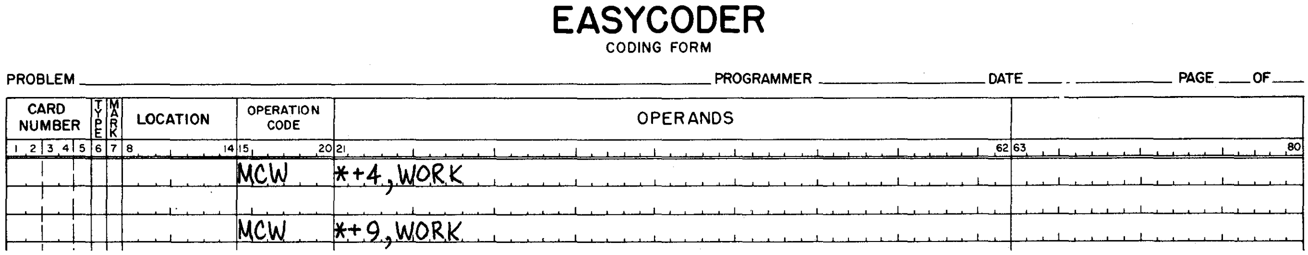
PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				TOTAL	A	FICA, TOTAX	
2							

The instruction shown above can be referred to elsewhere in the program via its tag (TOTAL). It should be noted, however, that this instruction is a valid statement only if the symbolic addresses FICA and TOTAX have been defined in the location field elsewhere in the source program.

SELF REFERENCE

It is sometimes convenient for an instruction to refer to itself. A self reference is indicated by an asterisk in the operands field of a source-program instruction. The assembler automatically replaces the asterisk with the address of the leftmost character of the instruction in which it appears. Address modification and relative addressing can be performed on asterisk operands.



EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ___ OF ___

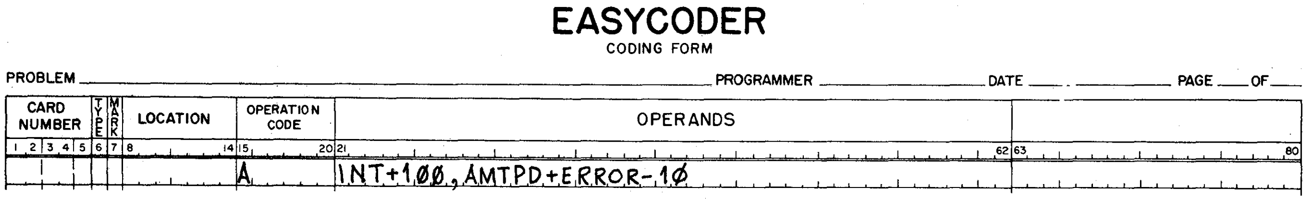
	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					MCW	*+4, WORK	
2							
3					MCW	*+9, WORK	

In the first sample entry above, the notation *+4 addresses the rightmost character of the instruction in which it appears (assuming that two-character address assembly has been specified). Since the function of this instruction is to move the field specified by the A address to that specified by the B address, the instruction itself will be moved to the field tagged WORK.

In the second entry, the notation *+9 refers to the rightmost character of the instruction stored immediately to the right of the MCW instruction (assuming that two-character address assembly has been specified). The instruction following the MCW instruction will be moved to the field tagged WORK when the MCW instruction is executed.

RELATIVE

Relative addressing, or address arithmetic as it is frequently called, can be used with all absolute addresses, symbolic addresses, and the self-reference symbol (*) (these three types of address codes are referred to as addressing "elements"). By using relative addressing, the programmer can refer to a source-program entry that is stored a specified number of locations away from a particular address. A relative address is specified by appending one or more address modifiers, each consisting of a sign and an addressing element, to another addressing element. The address modifier designates a memory location relative to the location specified by the basic addressing element. For example, the instruction below causes the contents of the field 100 characters beyond the field tagged INT to be added algebraically to the contents of the field 10 characters before the sum of the addresses defined by the tags AMTPD and ERROR.



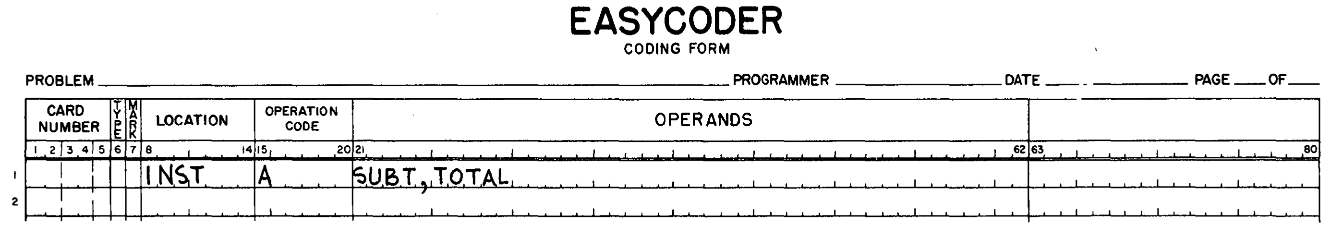
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				A	INT+1ØØ,AMTPD+ERROR-1Ø	

The number of symbolic tags required to write a program can be greatly reduced by the use of relative addressing. The programmer decides how many and which fields in a program to tag and which to reference by relative addressing.

A certain amount of caution is required in the use of relative addressing. First of all, relative addresses are not automatically corrected as a result of subsequent insertions or deletions in the source program. The programmer must remember to adjust manually the address modifiers affected by such changes. Secondly, if relative addressing is used to refer to an operand address in another instruction, care must be taken to insure that the address is referenced by its rightmost character. For example, the A address of the instruction shown below could be referred to elsewhere in the program as INST+2 or INST+3, depending on whether two- or three-character address assembly were specified.



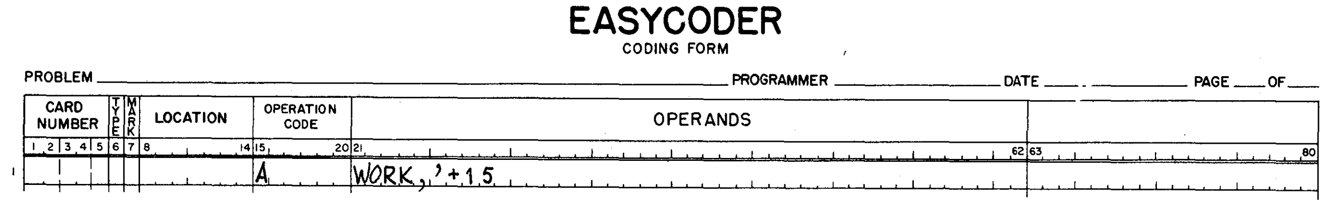
EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ___ OF ___

	CARD NUMBER 1-5	TYPE 6	MARK 7	LOCATION 8-14	OPERATION CODE 15-20	OPERANDS 21-62	63-80
1				INST	A	SUBT,TOTAL	
2							

OUT-OF-SEQUENCE

The valid address codes also include the special symbol apostrophe (printer '; keypunch 8, 2; octal 12). This symbol is an element whose value is equal to the current value of the out-of-sequence base (OSB). It is followed by an address modifier to specify the address of the desired operand. The OSB is set by means of the XBASE instruction (see page 7-18).



EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ___ OF ___

	CARD NUMBER 1-5	TYPE 6	MARK 7	LOCATION 8-14	OPERATION CODE 15-20	OPERANDS 21-62	63-80
1					A	WORK,'+15	

In the sample statement above, assume that the out-of-sequence base (OSB) has been set to 600 (by the XBASE instruction). The data in the field tagged WORK will be added to the data in the field whose rightmost location is 615 (600 + 15). The result will then be stored in the field whose rightmost location is 615.

BLANK

There are two conditions for which a blank operand field is valid:

1. The instruction does not require an operand (e. g. , the Halt and No Operation instructions).
2. The operands are implicitly addressed: the A operand is specified by the contents of the A-address register (AAR); the B operand is specified by the contents of the B-address register (BAR).

If either or both operand addresses are to be supplied by other instructions (as illustrated below in the description of address literals), the affected operands should be represented by zeros; they should not be left blank.

LITERALS[1]

The purpose of a literal is to allow the programmer to write in the operands field of a symbolic program statement the actual data (as opposed to the address of the field containing

[1] Not available with Easycoder A.

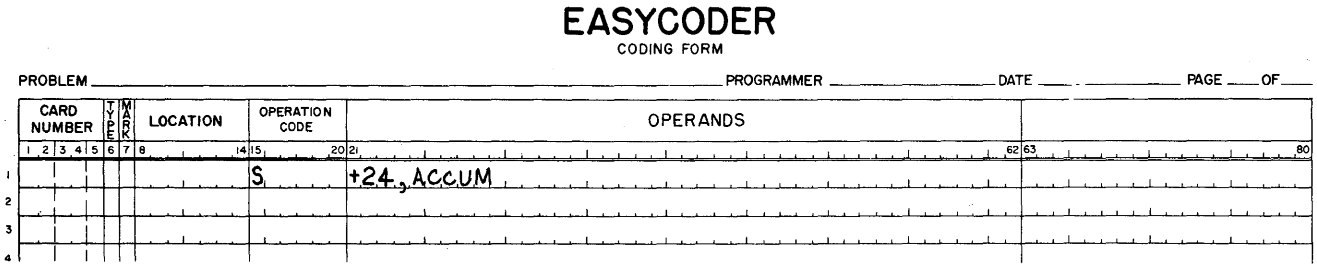
the data) to be operated on by an instruction. Easycoder B users can code all literals, except binary, with a maximum length of 40 characters; a binary literal can be coded with a maximum length of six characters. For users of Easycoder C or D, the maximum length of any literal can be 63 characters.

The assembler automatically assigns a storage field for each literal and inserts its address (i.e., the address of its rightmost character) in the operands field of the instruction in which it appears. In effect, for every literal appearing in the source program, the assembler generates a constant containing the value of the literal, with a word mark in the leftmost character position.

NOTE: If the constant generated from a literal occupies from one to five storage locations, it is assigned a storage address only once in the program, regardless of the number of times the literal appears in the source program. (For Easycoder C or D, the constant is assigned a storage address only once in the program if it occupies from one to six storage locations.) A constant that exceeds five characters (six for Easycoder C or D) is assigned a storage address each time the corresponding literal appears in the source program. The latter condition can be avoided by using a DCW statement (see page 6-2) whenever a long literal is to be used more than once in the source program.

Decimal Literals

Decimal literals are specified by writing a plus or minus sign followed by the value of the literal. When the literal is assigned to a storage field, the assembler places the sign in the zone bits of the units position of the resulting constant. Unsigned decimal values can be coded as alphanumeric literals.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					S	+24,ACCUM	
2							
3							
4							

The statement above illustrates the use of a decimal literal. The instruction causes the value 24 to be subtracted from the contents of the field tagged ACCUM.

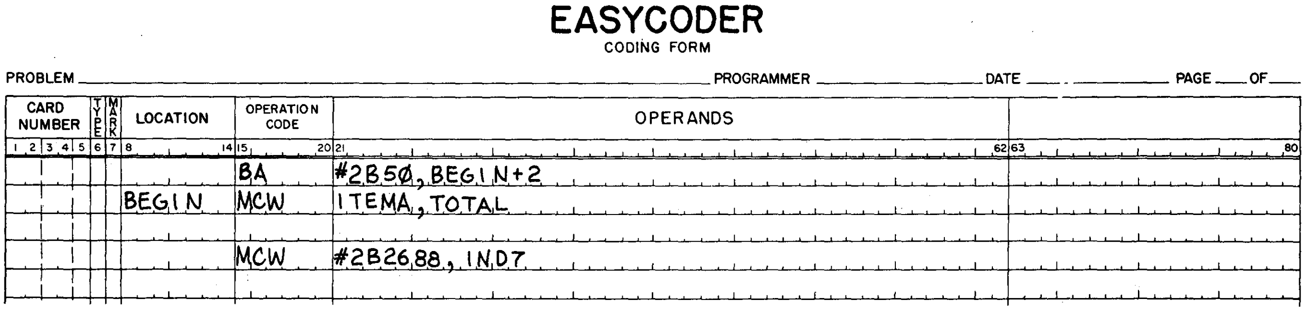
Binary Literals

A binary literal is represented as a decimal entry in the operands field of a symbolic instruction. The assembler automatically converts the decimal entry into a binary value and stores it (right-justified) in the storage field. The programmer must specify the number of six-bit characters used to store this value.

A binary literal is coded by writing a # sign, followed by a number which specifies how many six-bit characters should be used to store the resulting binary value, followed by the letter B, followed by the decimal representation of the desired binary literal.

NOTE: If the decimal representation of the binary literal is preceded by a minus sign, the assembler will store the binary literal in two's-complement form.

The first instruction below causes the binary equivalent of 50 (expressed as a continuous 12-bit binary value) to be added to the contents of the field tagged BEGIN+2. The second instruction has been included to illustrate how a binary literal can be used in address modification. In effect, the first instruction modifies the A address of the second instruction by a value of +50. The third instruction causes the binary equivalent of 2, 688 (expressed as a 12-bit binary value) to be moved to the field tagged IND7.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					BA	#2B5Ø,BEGIN+2	
2				BEGIN	MCW	ITEMA,TOTAL	
3							
4					MCW	#2B2688,IND7	
5							

Octal Literals

Octal literals are coded in octal notation (see Appendix A). The programmer must specify the number of six-bit characters required to store an octal literal.

NOTE: Since every octal digit can be represented as three bits, each six-bit character used to store an octal literal contains two octal digits. For example, an octal literal composed of eight octal digits can be stored in a four-character field.

An octal literal is coded in the same format as a binary literal except that the letter B used in the binary literal is replaced by the letter C. The constant stored by the assembler is always left-justified in the storage field.

EASYCODER
CODING FORM

PROBLEM PROGRAMMER DATE PAGE OF

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					HA	#3C7777,MASK	
2							

The A operand in the above statement is a four-digit octal literal. The assembler will store it left-justified in a three-character field, as 777700.

Alphanumeric Literals

An alphanumeric literal is specified by writing the @ symbol before and after the value of the literal. This type of literal can contain blanks, decimal, alphabetic, and special characters (excluding the @ symbol).

EASYCODER
CODING FORM

PROBLEM PROGRAMMER DATE PAGE OF

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					MCW	@ACCOUNTS PAYABLE,1Ø/19/65@,PRINT	

The statement above illustrates the use of an alphanumeric literal. The instruction causes the information contained within the @ symbols to be moved to the field tagged PRINT.

EASYCODER C AND D OPTIONS

In addition to the form specified above, users of Easycoder C or D have available to them three other methods of coding alphanumeric literals.

1. A number sign (#) is followed by a number from 1 through 63 which specifies the number of characters in the literal; this number is, in turn, followed by the letter A and the literal.

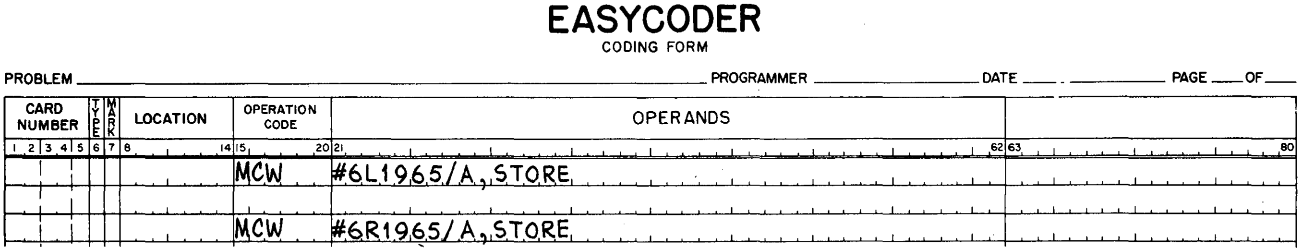
EASYCODER
CODING FORM

PROBLEM PROGRAMMER DATE PAGE OF

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					MCW	#14A6 LBS @ 21¢/LB,PRINT	

In the above example there are 14 characters in the literal. The instruction causes these 14 characters to be moved to the field tagged PRINT.

2. If it is desired to set an item mark (in addition to a word mark) in the leftmost position of the literal constant field, a number sign (#) is followed by a number from 1 through 63 which specifies the number of characters in the literal; following this number is the letter L and the literal (see the first example below).



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

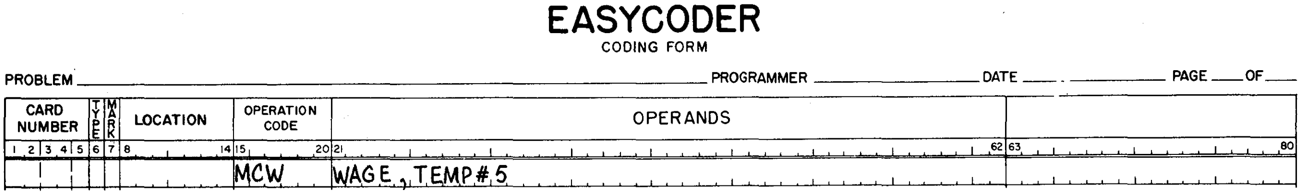
CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				MCW	#6L1965/A,STORE	
2						
3				MCW	#6R1965/A,STORE	

3. If it is desired to set an item mark in the rightmost position of the literal constant field, a number sign (#) is followed by a number from 1 through 63 which specifies the number of characters in the literal; following this number is the letter R and the literal (see the second example above).

NOTE: In form (1), alphanumeric literals of six characters or less are stored in a literal table and duplicates are eliminated. The duplicates are not, however, eliminated in forms (2) and (3).

Area Defining Literals

An area defining literal may be used to define and reserve a working area in memory without using a separate data formatting instruction. The address which defines the area is written as a symbolic tag. The size of the area to which the literal address refers is specified as a decimal value following the literal address and separated from it by a # symbol.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

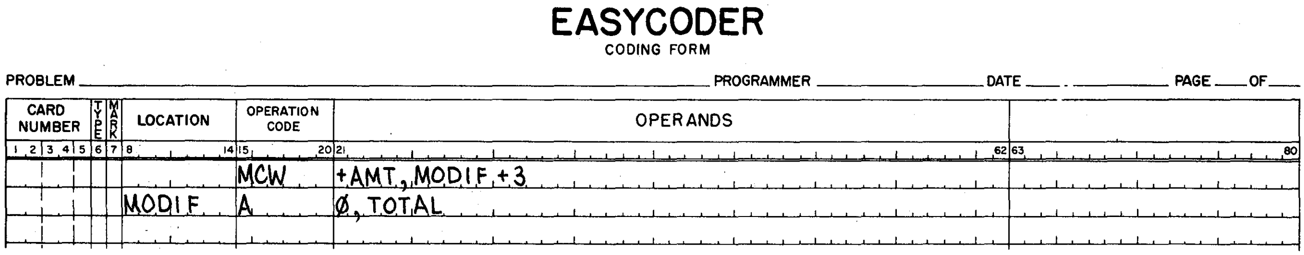
CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				MCW	WAGE,TEMP#5	

In the instruction above, the entry TEMP#5 causes the assembler to reserve a blank five-character area with a word mark set in the leftmost character position. The address of the rightmost character in this area is assigned to the tag TEMP. Therefore, TEMP can be used as a symbolic address elsewhere in the source program, because both the tag and size of the area to which it refers are defined. The sample instruction causes the contents of the field tagged WAGE to be moved to the field tagged TEMP.

Address Literals

An address literal enables the programmer to specify a symbolic address in the operands field of an instruction such that the assembler will use the address as an operand. A symbolic address can be used as an address literal only if it is defined elsewhere in the symbolic program. The tag used as an address literal must be preceded by a plus sign. The length of the address is determined by the current addressing mode (the defined address can be two, three, or four characters long).

An address literal (+AMT) is used in the first sample entry below. Assume that AMT has been defined elsewhere in the program and has been assigned an absolute address of 800. The absolute address of AMT, as opposed to the contents of the field tagged AMT, replaces the address literal. The first instruction below causes the value 800 (the absolute address assigned to AMT) to be moved to an address three greater than the location tagged MODIF. The second entry shows how an operand address can be supplied by another instruction. Specifically, the absolute address assigned to the tag AMT is supplied as the A address of the instruction tagged MODIF. This instruction causes the contents of the field tagged AMT (i. e., the field whose rightmost character is stored in location 800) to be added algebraically to the contents of the field tagged TOTAL.



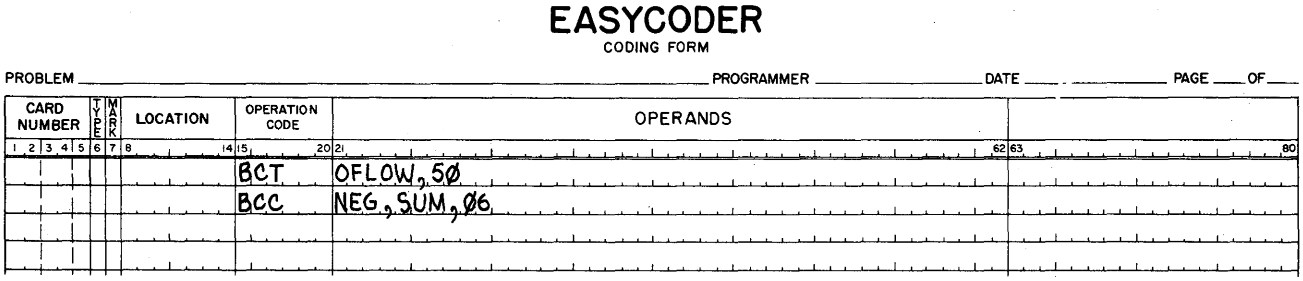
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER 1-5	TYPE 6	MARK 7	LOCATION 8-14	OPERATION CODE 15-20	OPERANDS 21-62	63-80
1					MCW	+AMT, MODIF+3	
2				MODIF	A	Ø, TOTAL	
3							

VARIANT CHARACTER

A variant character can be expressed as one alphanumeric character, as two octal digits, or as a symbolic tag.[1] It is written following the operand entries and separated from the last entry by a comma. Octal representation of valid characters are listed in Appendix B.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER 1-5	TYPE 6	MARK 7	LOCATION 8-14	OPERATION CODE 15-20	OPERANDS 21-62	63-80
1					BCT	OFLOW, 5Ø	
2					BCC	NEG, SUM, Ø6	
3							
4							

The first instruction above tests an indicator specified by the variant character. If the indicator is on, the instruction causes the program to branch to the address tagged OFLOW. As might be expected, the octal digits 50 represent the overflow indicator. The second instruction causes the single character at the location tagged SUM to be examined for a particular bit

[1] A symbolic tag, composed of at least two characters, may be used to represent (1) a variant character, or (2) a group of input/output control characters. The number of I/O control characters that may be represented varies from one to six (using either Easycoder A or B) or from one to four (using other Easycoder C or D). The symbolic tag must be defined before it is used in the input/output instruction; the Control Equals statement (CEQU) is generally used for this purpose (see page 7-13).

configuration as specified by the variant. In this case the variant 06 specifies that the character should be examined for a negative sign. If the desired bit configuration is present, the program branches to the address tagged NEG.

INPUT/OUTPUT CONTROL CHARACTERS

Input/output control characters can be used only in conjunction with input/output instructions (see Section VIII). One or more of these characters may be written following the A-address entry in an input/output instruction, each preceded by a comma. Input/output control characters may be coded as single alphanumeric characters, as pairs of octal digits, or as symbolic tags. [1]

ADDRESS MODIFICATION CODES

In a system equipped with the Advanced Programming Instructions (Feature 010 or 011 in Model 200; standard in Models 1200, 1250, 2200, and 4200), two address modification codes are valid in the operands field of a source-program statement: indexed and indirect. These codes allow the modification of operand addresses without altering the instructions in which the addresses appear. This is in direct contrast to the permanent alteration of an instruction that results from using a binary arithmetic instruction to modify either or both operand addresses.

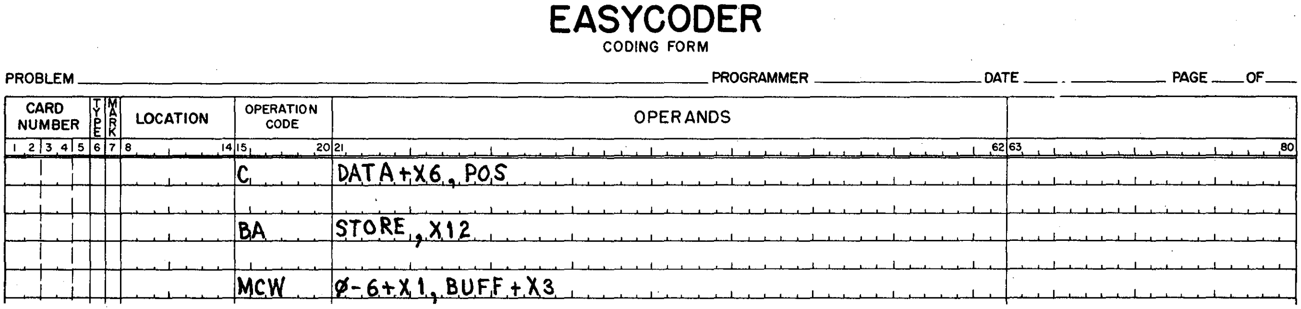
INDEXED

Indexed addressing is performed by appending to the address being modified a code to indicate which of the index registers is to be used. The code consists of a plus sign followed by an X or Y and a decimal number from 1 to 15. [2]

If an index register is to be specified in the operands field of an instruction for other than indexing purposes, it is referred to by its absolute address rather than its symbolic address. For instance, absolute address 24 is used instead of the corresponding symbolic address X6. However, the programmer may use the symbolic address if he equates it to the absolute address using an EQU statement (see page 7-12).

[1] See footnote, page 5-20.

[2] Figure 4-3, page 4-9, pictures the possible locations of Series 200 index registers. Table 4-1, page 4-10, indicates the number of index registers simultaneously available to a program. Tables 4-2 and 4-3 on pages 4-12 and 4-14, respectively, indicate the address modifier and absolute locations corresponding to each symbolic index-register address. The number of index registers which can be referenced sumbolically also depends on the assembler being used, as described on page 6-8.



EASYCODER

CODING FORM

PROBLEM ________ PROGRAMMER ________ DATE ____ . ________ PAGE ____ OF ____

CARD NUMBER 1 2 3 4 5	TYPE 6	MARK 7	LOCATION 8 14	OPERATION CODE 15 20	OPERANDS 21 62	63 80
1				C	DATA+X6,POS	
2						
3				BA	STORE,X12	
4						
5				MCW	Ø-6+X1,BUFF+X3	

The first instruction above causes the contents of the field designated by the tag DATA as modified by the contents of index register X6 to be compared to the contents of the field tagged POS. The second instruction causes the contents of the field tagged STORE to be added (in binary) to the contents of index register X12. The use of the symbolic designation X12 implies that an EQU statement was used to equate it to the absolute address of index register X12. The third instruction illustrates how an indexed address can be coded to generate an effective address which is less than the value stored in the specified index register. The zero is used because an operand address cannot be introduced with a plus or a minus sign. Thus, the effective A address of the MCW instruction will be a value six less than that stored in index register X1 (i. e., if index register X1 contains 126, the effective A address is 120).

Three- or four-character address assembly must be specified (see ADMODE, page 7-11) whenever indexed addressing is to be performed. When the assembler translates an indexed address into a machine-language entry (see Figures 5-6 and 5-7), the translated index register designator is automatically inserted into the address modifier bits of the assembled address.

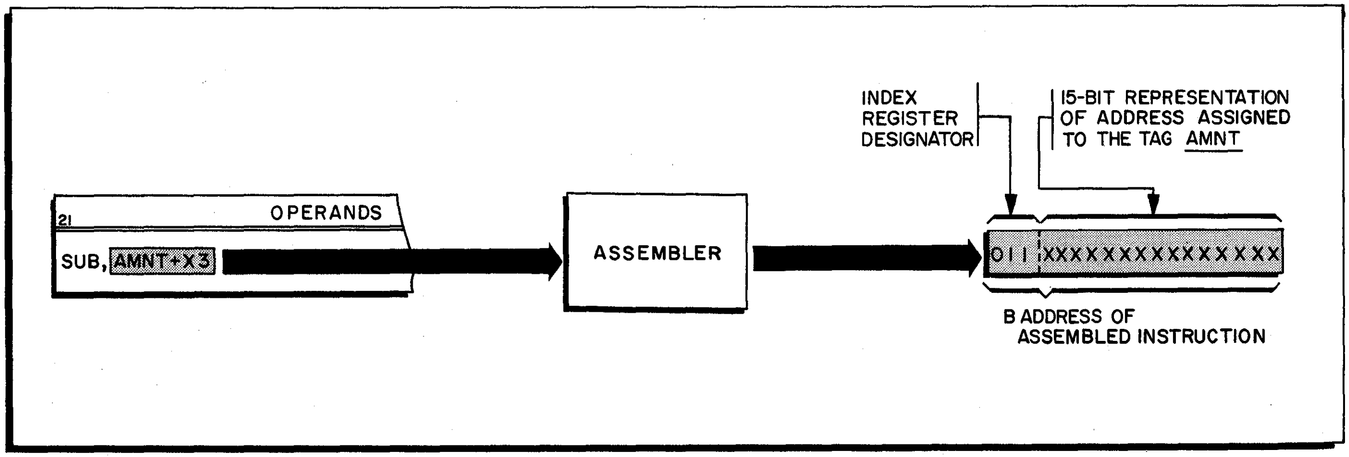

INDEX REGISTER DESIGNATOR
15-BIT REPRESENTATION OF ADDRESS ASSIGNED TO THE TAG AMNT
21
OPERANDS
SUB,AMNT+X3
ASSEMBLER
011 XXXXXXXXXXXXXXX
B ADDRESS OF ASSEMBLED INSTRUCTION

Figure 5-6. Assembly of Indexed Address in Three-Character Addressing Mode

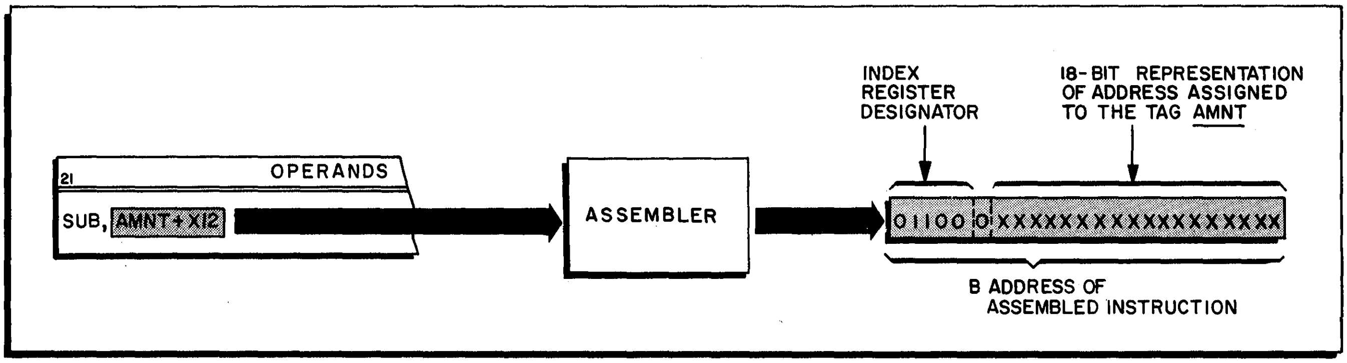
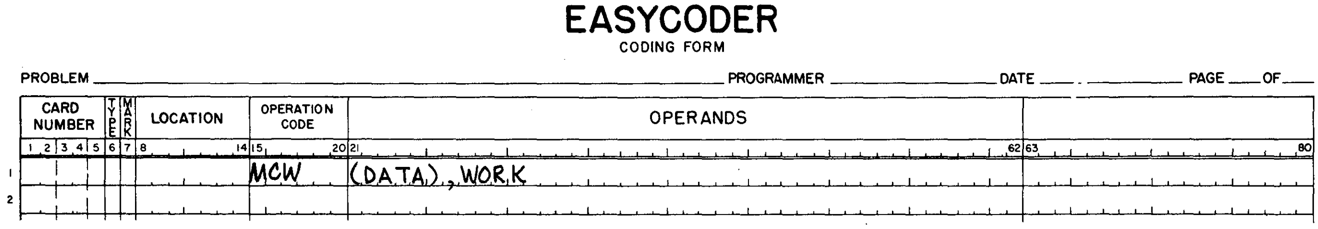


Figure 5-7. Assembly of Indexed Address in Four-Character Addressing Mode

INDIRECT

An indirect address is specified by enclosing the address (either symbolic or absolute) in parentheses.[1] For example, in the sample instruction below, the parentheses around the tag DATA indicate to the assembler that DATA refers to the leftmost character of a field containing another address. This second address may be a direct, an indexed, or another indirect address. If it is direct or indexed, it specifies the rightmost character of a data field. If it is indirect, it specifies the leftmost character of a field containing another address.



EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ____ OF ____

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					MCW	(DATA),WORK	
2							

Three- or four-character address assembly must be specified whenever indirect addressing is to be used. When assembler translates an indirect address into a machine-language entry (see Figures 5-8 and 5-9), a binary value of 111 (three-character mode) or 10000 (four-character mode) is automatically inserted into the address modifier bits of the assembled address.

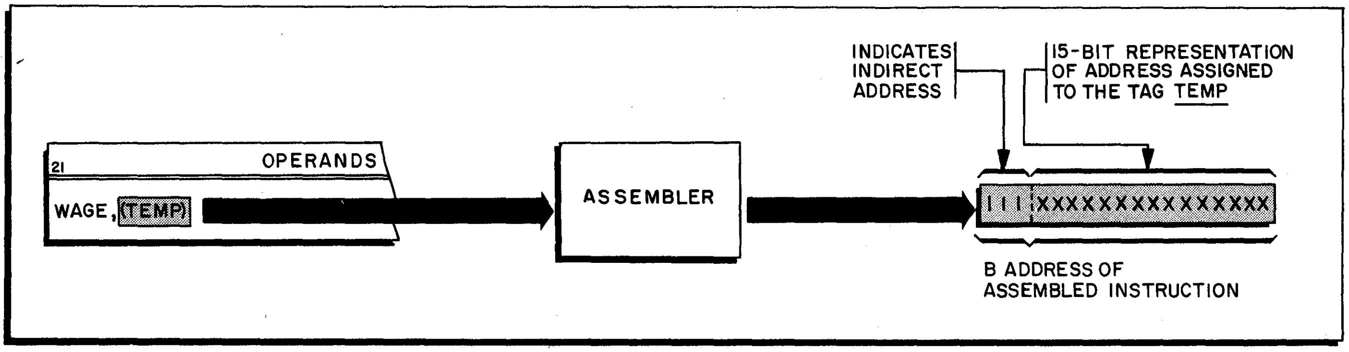


Figure 5-8. Assembly of Indirect Address in Three-Character Addressing Mode

[1] The left parenthesis corresponds to keypunch symbol % (card code 0, 8, 4), the right parenthesis to keypunch symbol ⌑ (card code R, 8, 4).

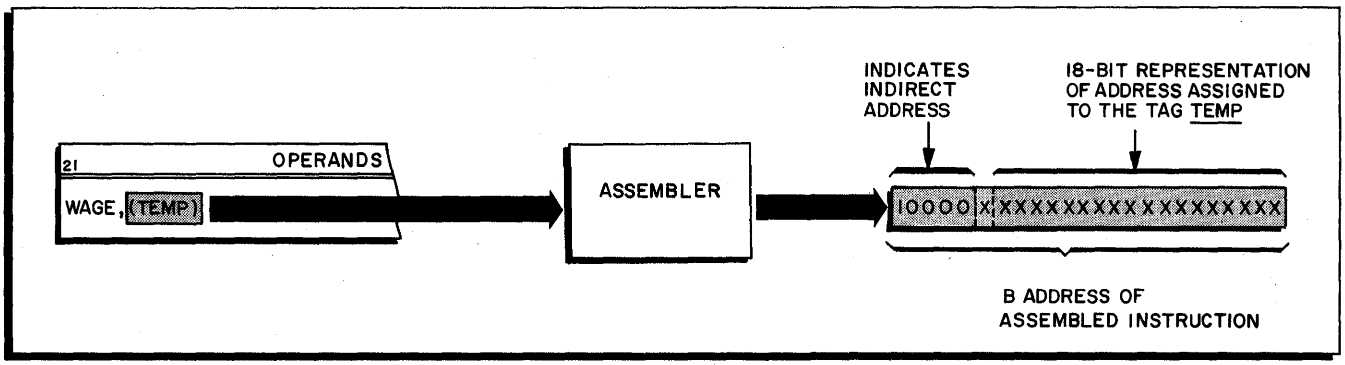


Figure 5-9. Assembly of Indirect Address in Four-Character Addressing Mode

6 DATA FORMATTING STATEMENTS

INTRODUCTION

A value or quantity which must remain fixed or which must be used repeatedly in a program is called a constant. A work area is an area in memory which is reserved for input data, cumulative processing, or output data. By employing data formatting statements, constants can be stored and work areas can be reserved without regard to their actual locations in memory. For instance, the programmer can use a data formatting statement to reserve an 80-character card input area and assign it a symbolic address such as CARDIN, without knowing the actual address of the field. Similarly, a data formatting statement makes it possible to store a constant, such as 2000, and to refer to it by a symbolic tag, such as CON3, without regard to the address at which the constant is stored. Table 6-1 lists the five data formatting statements used with Easycoder symbolic language.

Table 6-1. Data Formatting Statements

Mnemonic Operation Code	Function
DCW	Define Constant with Word Mark
DC	Define Constant without Word Mark
RESV	Reserve Area
DSA	Define Symbol Address
DA	Define Area*

*NOTE: The Define Area statement cannot be employed with the Easycoder A Assembly System.

Although data formatting statements are coded in the same format as most symbolic machine instructions (data processing statements), they are not treated as instructions by an assembler. Instead they are treated as definitions which cause the assembler to perform certain activities but which are not executed during a program run. Since data formatting statements are not executed during a program run, they should not be written in the body of the symbolic program.

Define Constant with Word Mark - DCW

By use of the DCW statement, a constant can be automatically stored in a field reserved by the assembler. In storing the constant, the assembler automatically sets a word mark in the leftmost character position of the storage field. Item marking may be specified as in Table 5-1 (page5-7). An L in column 7 thus results in a record mark with a DCW statement.

NOTE: If Easycoder C or D is being used, and if unusual high- and low-order punctuation is required, the programmer may use a set II punctuation indicator as shown in Table 5-2 (page 5-7).

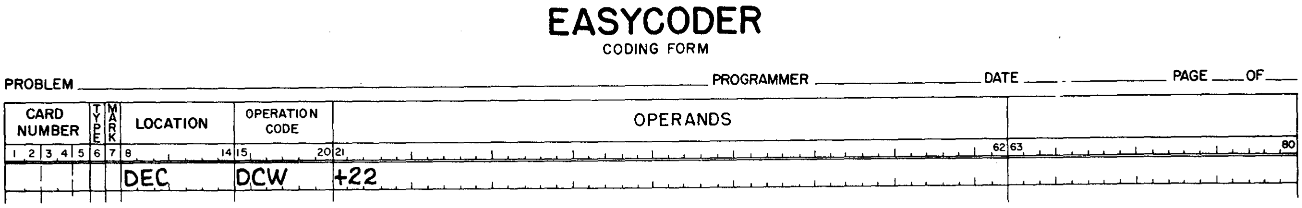
The constant can be assigned a tag. If the tag is left-justified in the location field, it is assigned to the address of the rightmost character of the constant. If the tag is indented one column, it is assigned to the address of the leftmost character of the constant.

NUMERIC CONSTANTS

Numeric constants may take any one of three forms: binary, octal, or decimal. For Easycoder A and B, octal and decimal constants can be coded with a maximum length of 40 characters, while the coding associated with a binary constant is limited to a maximum of six characters. However, for Easycoder C and D, the maximum length of the storage field which can be occupied by a numeric constant is 63 characters.

Decimal Constants

Signed decimal constants are specified by writing a plus or a minus sign in the first column of the operands field, followed by the value of the constant. When the constant is assigned to a storage field, the assembler places the sign in the zone bits of the rightmost character of the constant.[1] Unsigned decimal constants are written left-justified in the operands field.



EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE __ OF __

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
			DEC	DCW	+22	

The statement above shows the decimal value of +22 defined as a decimal constant.

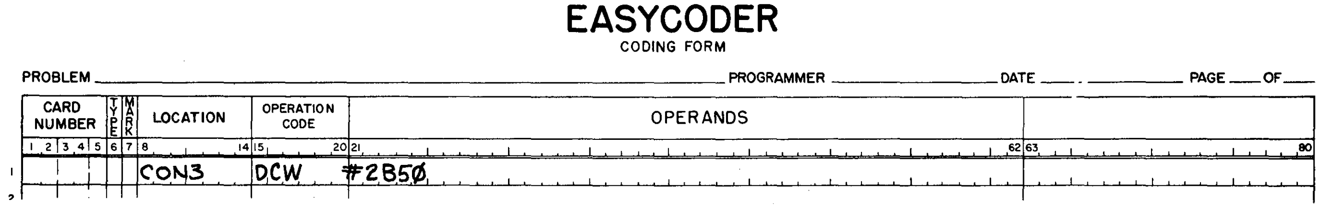
Binary Constants

A binary constant is actually written as a decimal entry (maximum value of 999999) which

[1] See the description of sign codes beginning on page 8-7.

is then automatically converted to a binary value by the assembler. The binary value is stored (right-justified) in the constant field.

To code a binary constant the programmer writes the following: (1) a # sign (in the first column of the operands field); (2) for Easycoder A or B, a number from 1 to 6 which designates the number of six-bit characters needed to store the resulting binary value (for Easycoder C or D, a number from 1 to 63); (3) the letter B; and (4) the decimal representation of the desired binary constant. Note that if the decimal representation of the binary constant is preceded by a minus sign, the assembler stores the binary constant in twos-complement form.



EASYCODER
CODING FORM

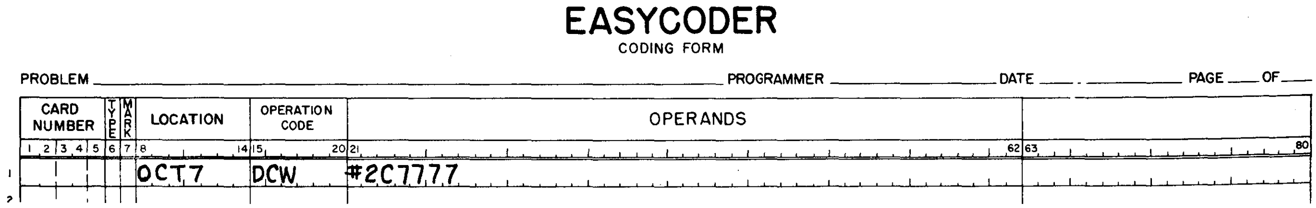
PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
1			CON3	DCW	#2B5Ø	
2						

The statement above shows the binary equivalent of 50 defined as a binary constant to be stored in two consecutive character locations.

Octal Constants

Octal constants are coded in octal notation (see Appendix A). To code an octal constant the programmer writes the following: (1) a # sign (in the first column of the operands field); (2) a number (not to exceed 20 for Easycoder A and B; not to exceed 63 for Easycoder C and D), which specifies the number of six-bit characters required to store the octal constant;[1] (3) the letter C; (4) the constant value. Note that the value stored by the assembler is always left-justified in the storage field.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
1			OCT7	DCW	#2C7777	
2						

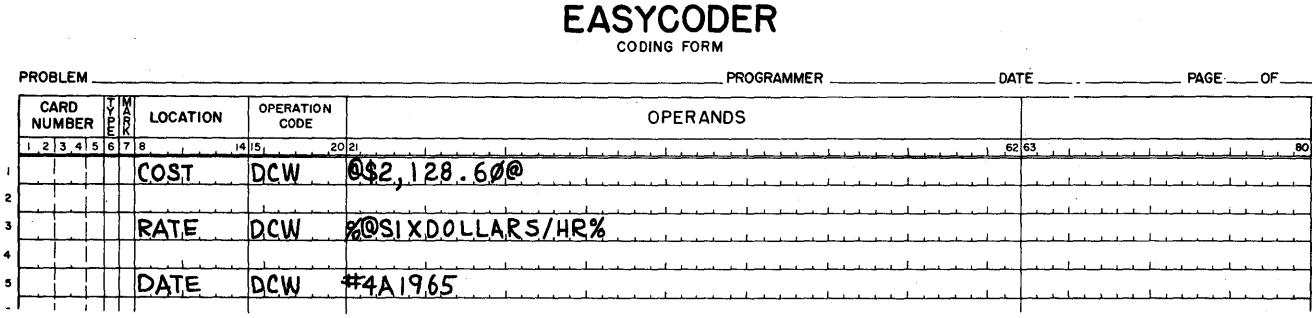
In the statement above, the octal value of 7777 is shown defined as an octal constant to be stored in two consecutive character locations.

[1] Recall that an octal digit can be represented as three bits; thus each six-bit character used to store an octal constant contains two octal digits. For example, an octal constant composed of six octal digits can be stored in a three-character field.

ALPHANUMERIC CONSTANTS

Alphanumeric constants may be coded in one of three ways:

1. Constants (including special symbols and blanks) may be written with the constant value enclosed in @ symbols (see the first entry below).
2. If the @ symbol is required in the constant, this constant is enclosed in any unused character other than blank, +, -, # (and F, for Easycoder D) or the digits 0 through 9 (see the second entry below).
3. A number sign (#) is followed by a number from 1 through 56 which specifies the number of alphanumeric characters contained in the constant; this number is, in turn, followed by the letter A and the alphanumeric constant (see the third entry below).[1]



EASYCODER
CODING FORM

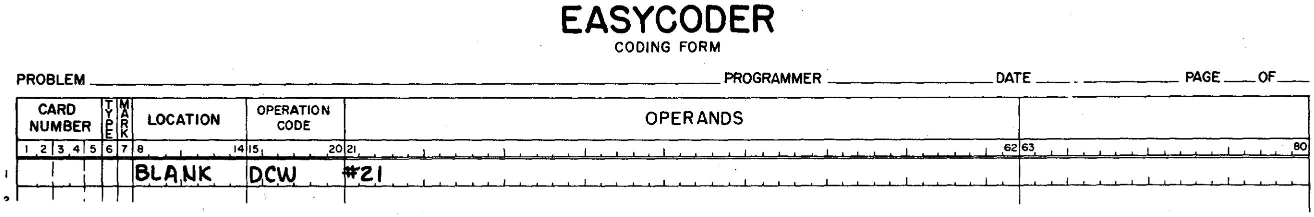
PROBLEM ________ PROGRAMMER ________ DATE ________ PAGE ___ OF ___

	CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
1				COST	DCW	@$2,128.6Ø@	
2							
3				RATE	DCW	%@SIXDOLLARS/HR%	
4							
5				DATE	DCW	#4A1965	

NOTE: The maximum number of alphanumeric characters which can be contained in the constant, of course, depends on the number of card columns available in the operands field. Thus it should be remembered that methods 1 and 2, above, require two card columns to format the constant, while method 3 requires either three or four columns.

BLANK CONSTANTS

The DCW statement may be used to reserve a field of blanks with a word mark in the leftmost character position of the field. The programmer writes a # symbol (in the first column) followed by a decimal value (from 1 to 40 for Easycoder A or B, from 1 to 63 for Easycoder C or D) which indicates the number of blank storage positions desired.



EASYCODER
CODING FORM

PROBLEM ________ PROGRAMMER ________ DATE ________ PAGE ___ OF ___

	CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
1				BLANK	DCW	#21	

[1] This third method of coding alphanumeric constants is applicable only when using Easycoder C or D.

The DCW statement above defines a 21-character blank field. The address assigned to this field by the assembler will be inserted in an object-program instruction whenever the tag BLANK appears in another symbolic-program entry.

FLOATING-POINT CONSTANTS

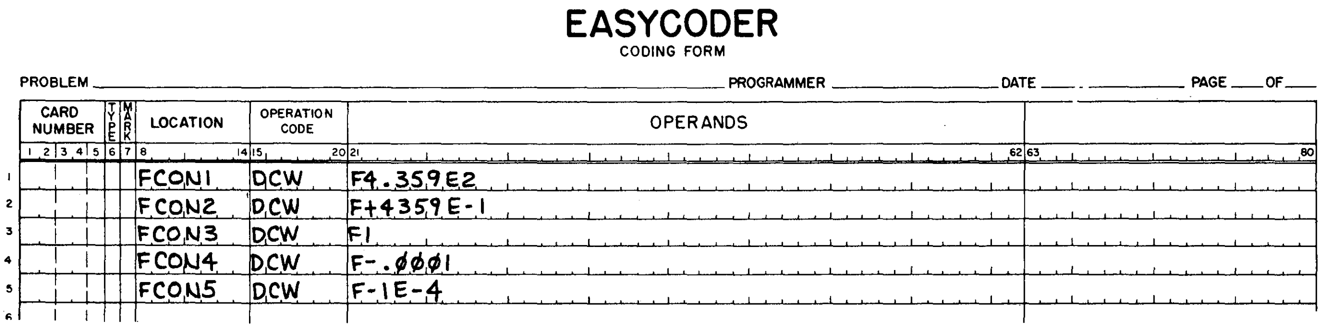
A floating-point constant is written as a decimal entry which is then automatically converted by the assembler to a fixed-length floating-point value, viz., a six-character binary mantissa followed by a two-character power-of-two exponent.

To code a floating-point constant the programmer writes the following:

1. The letter F.
2. A decimal number, the mantissa which may be signed or unsigned and which may contain a maximum of 11 digits with or without a decimal point.
3. The letter E.
4. A decimal number, the exponent, which must be between 0 and 616, inclusive, and may be signed or unsigned.

If an exponent of zero is desired, the letter E and the decimal number which follows it are not required.

NOTE: If the mantissa and/or the exponent is preceded by a minus sign, the assembler stores the corresponding value in twos-complement form.



EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ____ OF ____

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				FCON1	DCW	F4.359E2	
2				FCON2	DCW	F+4359E-1	
3				FCON3	DCW	F1	
4				FCON4	DCW	F-.ØØØ1	
5				FCON5	DCW	F-1E-4	
6							

The first two entries above (FCON1 and FCON2) result in the same floating-point value when converted by the assembler. FCON1 uses a decimal point while FCON2 arrives at the same result by using a negative exponent. This is also true for FCON4 and FCON5.

Define Constant - DC

The DC statement is functionally the same as the DCW statement, the only exception being the absence of automatic word marking. This statement may thus be used in place of the DCW statement if a constant is to be stored without a word mark in its leftmost character position. The programmer, however, may still specify item marking as shown in Table 5-1 (page 5-7).

NOTE: If Easycoder C or D is being used, and if unusual high- and low-order punctuation is required, the programmer may use a set II punctuation indicator as shown in Table 5-2 (page 5-7).

Reserve Area - RESV

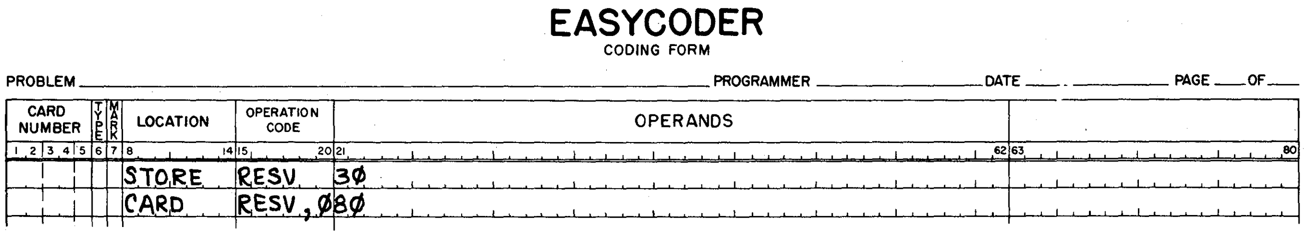
Use of the RESV statement enables the programmer to reserve an area of memory. Unlike the DC and DCW statements (which cause data to be loaded into an area reserved by the assembler), the RESV statement does not normally alter the contents of the area defined. Rather, it simply sets aside a storage area to which the programmer can refer by a symbolic tag. The reserved area can be cleared to zeros by means of the CLEAR statement (see page 7-19). The number of characters in the reserved area must be specified in the operands field of the RESV statement.

NOTE: When used with Easycoder A or B, the RESV statement must contain a nonzero value in the operands field.

A symbolic tag may be written in the location field. If the tag is left-justified, it is assigned to the rightmost location of the reserved area. If the tag is indented one column, it is assigned to the leftmost location of the reserved area.

When used with Easycoder C or D, the RESV statement can not only reserve a specified area but can also load that area with a particular character. The character to be loaded into each location of the reserved area is coded in the op code field immediately following a comma and the mnemonic code. If the mnemonic RESV is followed only by a comma, the reserved area is cleared to blanks.

NOTE: There is no automatic word marking for the reserved areas, nor may column 7 of the RESV statement be used with Easycoder A or B to set punctuation. However, if Easycoder C or D is being used, the programmer may use a set I or II punctuation indicator (see page 5-7).



EASYCODER

CODING FORM

PROBLEM ________ PROGRAMMER ________ DATE ________ PAGE ____ OF ____

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				STORE	RESV	3Ø	
2				CARD	RESV,Ø	8Ø	

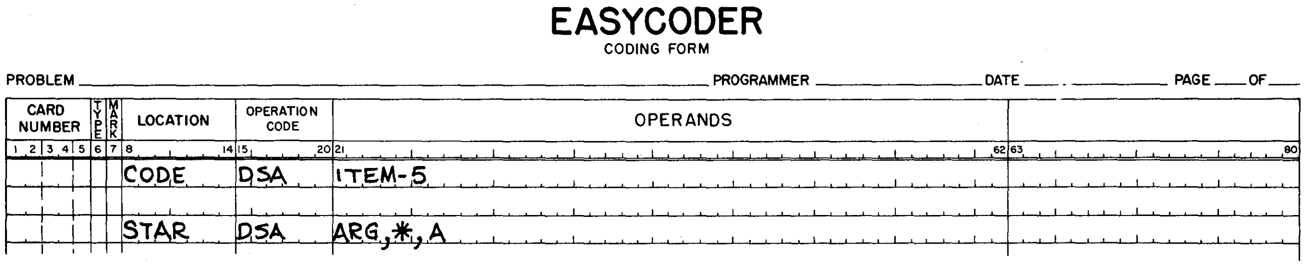
The first statement above reserves 30 consecutive character positions that can be addressed via the tag STORE. Note that by referring to the reserved area via a symbolic tag, the programmer need not know its actual location in memory. The second RESV statement, assembled by Easycoder C or D, reserves 80 consecutive locations and clears the reserved area to zeros.

Define Symbol Address - DSA

The DSA statement can be used to store one or two addresses, or two addresses and a variant character, as a constant. Any valid address can be stored as a constant; the length of each address is determined by the current addressing mode (each address will be two, three, or four characters long).

An item mark may be specified as shown in punctuation set I, page 5-7. In addition, the DSA statement automatically places a word mark in the leftmost character position of the constant (thus an L in column 7 results in a record mark in this position).

NOTE: If Easycoder C or D is being used, and if unusual high- and low-order punctuation is required, the programmer may use a set II punctuation indicator as shown in Table 5-2 (page 5-7).



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				CODE	DSA	ITEM-5	
2							
3				STAR	DSA	ARG,*,A	

The first statement above permits the address of the field five characters before the field tagged ITEM to be referred to in the program by the tag CODE.

The second statement allows the stored constant consisting of the address assigned to ARG, the address assigned to the self-reference indicator *, and the variant character A (i.e., octal 21) to be referred to by the tag STAR.

Define Area - DA[1]

A specified area within the main memory can be defined and reserved by using the DA statement. In addition to defining an area, the DA statement can also define fields and subfields within the reserved area. This statement can also define two or more contiguous areas if these areas are identical in format. In other words, the programmer uses a DA statement to provide the assembler with the following basic information:

1. The number (n) and size (s) of the reserved area(s). (Both n and s can be represented by numbers up to 4,095, depending upon the amount of memory available.)
2. The index register (Xm or Ym) to be associated with each reference to a field or subfield within the reserved area(s) (optional).

[1] The Define Area statement can not be employed with the Easycoder A Assembly System.

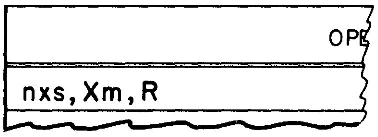
3. The character R which will place a record mark one position to the right of the rightmost reserved area (optional).

   NOTE: Additional parameters may be employed with Easycoder C and D (see page 6-10).

A DA statement consists of a heading line which defines an area(s), plus one or more subsequent lines of coding which defines the fields and subfields within the area(s). The heading line can contain a symbolic tag in the location field. If this tag begins in column 8, it refers to the rightmost location of the entire area, exclusive of the record mark (if present); if the tag starts in column 9, it refers to the leftmost location of the entire area. Item marks may be specified in column 7 of the heading line by using set I punctuation indicators as shown in Table 5-1 (page 5-7).

NOTE: The list of punctuation indicators specified in set II (page 5-7) cannot be used with DA statements.

The operands field in the heading line has the following format:



nxs,Xm,R

If a single 80-character area is to be defined, the value of nxs is 1x80. If four identical 80-character areas are to be defined, the value of nxs is 4x80.

The DA statement can be indexed by writing an index register designator (from X1 through X15 or from Y1 through Y15)[1] following the area definition. All references to the field and subfields defined in the DA statement will be automatically indexed by the specified index register, but references to the tag assigned to the entire area will not be indexed. For example, the statement on the next page indicates that all references to the fields and subfields in the 113-character area tagged BUFFER will be indexed by the index register X2; references to the tag BUFFER, however, will not be indexed.

Note that the area definition nxs does not include an allowance for the character position containing the record mark, although this position (if any) is also reserved. For example 4x80 will cause 320 character positions to be reserved. If a record mark is placed one position to the right of the last area, a total of 321 character positions is reserved.

The index register applied to a field or subfield can be changed from that specified in the DA statement by designating a different register in the operands field of an instruction which

[1] Index registers X1 through X6 are used with Easycoder B, while index registers X1 through X15 and Y1 through Y15 can be used with Easycoder C or D.

references the field or subfield. The effect of indexing on a field or subfield can be cancelled by writing X0 as the index register designator in the references in which indexing is not wanted.

As stated above, the heading line may be followed by one or more lines of coding which define fields and subfields within the reserved area(s). As many of these lines as necessary may be used, and these fields and subfields may be defined in any order desired. Positions within each reserved area are numbered sequentially from left to right, starting with one. The coding lines which define fields and subfields must have blank op code fields; each such line may contain a symbolic tag in the location field, if desired.

Fields and subfields are specified as follows:

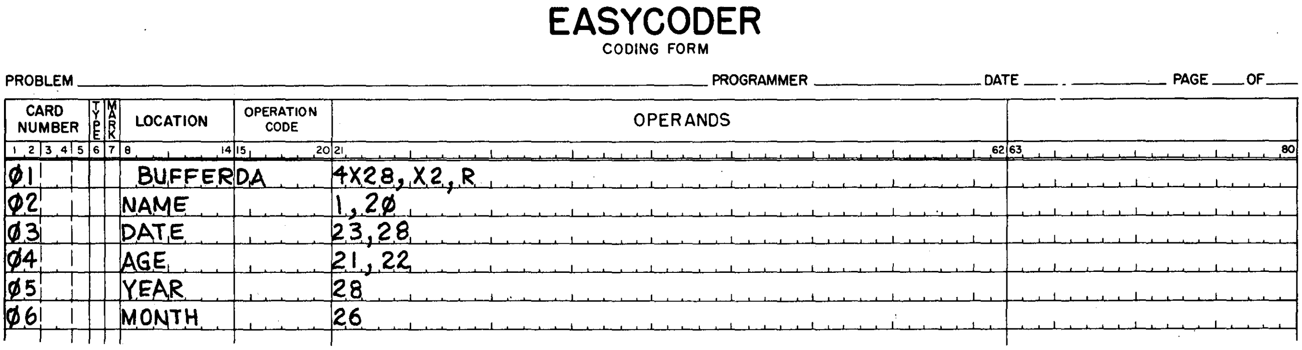
Fields: The lowest and highest positions of the field are written in that order in the operands field, separated by a comma. (If a one-character field is desired, its position number must be written twice in the operands field, separated by a comma.) A word mark is automatically placed in the leftmost position of the field in memory. Item marks may be specified as shown in Table 5-1 (page 5-7).

Subfields: For a subfield, only the rightmost position is specified. Word marks are not set; however, item marks may be specified as shown in Table 5-1 (page 5-7).

NOTE: The list of punctuation indicators specified in set II (page 5-7) can not be used with DA statements.

The assembler does not normally clear the defined area. However, the programmer has the option of clearing the area to a specified character by placing a comma and the desired character after the mnemonic code DA in the op code field. The presence of only a comma after the op code implies that the area will be cleared to blanks. When the defined area is cleared, all punctuation is also cleared before setting the "field" punctuation.

The sample coding below illustrates what a DA statement might look like.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1	Ø1			BUFFER	DA	4X28, X2, R	
2	Ø2			NAME		1, 2Ø	
3	Ø3			DATE		23, 28	
4	Ø4			AGE		21, 22	
5	Ø5			YEAR		28	
6	Ø6			MONTH		26	

The heading line specifies the following information:

1. Four consecutive, identical areas, each 28 characters long, will be reserved.
2. The tags NAME, DATE, AGE, YEAR, and MONTH, when referred to in symbolic instructions, will be indexed by index register X2.
3. A record mark will be set in the rightmost character position of the entire 113-character reserved area.
4. The entire 113-character area can be referred to via the tag BUFFER. (This tag refers to the leftmost position of the area because it is indented. It is not automatically indexed by index register X2.)

Lines two, three, and four define fields. Word marks will be set in positions 1, 21, and 23 in each of the four identical areas. Lines five and six define subfields: position 28 indicates the year within the date, while position 26 indicates the month within the date.

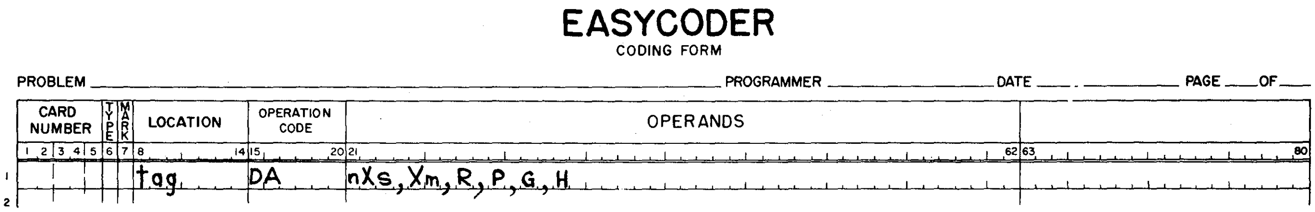
EASYCODER C AND D OPTIONS

When used with Easycoder C or D, the DA statement may make use of the following parameters (in addition to the n, s, Xm, and R parameters specified on page 6-8).

1. The character P: Coding this character in the heading line of a DA statement causes the special character 72_8, together with an item mark, to be placed at the end of each area as an additional character.
2. The character G: Coding this character in the heading line causes the special character 32_8, together with a record mark, to be placed one position to the right of the last area.
3. The character H: Coding this character in the heading line instructs the assembler to associate the index register (Xm or Ym) with each reference to the tag in the location field of the DA statement, as well as with each reference to a field or subfield within the reserved area(s).

   NOTE: If a symbolic tag is used, it is not automatically indexed by the specified index register (Xm or Ym) unless parameter H is employed. This parameter is meaningless if no index register is specified.

The format of a DA statement heading line employing all parameters is illustrated below.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER 1-5	TYPE 6	MARK 7	LOCATION 8-14	OPERATION CODE 15-20	OPERANDS 21-62	63-80
			tag	DA	nXs,Xm,R,P,G,H	

7

ASSEMBLY CONTROL STATEMENTS

INTRODUCTION

Assembly control statements provide programmer control over the assembly of the source program. These statements resemble data formatting statements in that they are treated as definitions. They control such functions as the addressing mode to be used in assembling specified instructions, the assignment of absolute locations to symbolic tags, etc. Used only during the assembly process, assembly control statements are never executed as instructions in the object program. The precise function of each assembly control statement depends upon the assembly system employed.

A summary of the assembly control statements available with Easycoder A, B, C, and D, together with the page where each statement is defined, may be found in Table 7-1. In addition, the heading of each statement in this section includes a table which indicates the assembly systems that may use that particular statement.

Table 7-1. Assembly Control Statements

Easycoder A		Easycoder B		Easycoder C		Easycoder D	
Assembly Control Statements	Page Ref.	Assembly Control Statements	Page Ref.	Assembly Control Statements	Page Ref.	Assembly Control Statements	Page Ref.
Program Header	7-22	Program Header	7-2	Program Header	7-3	Program Header	7-3
				Segment Header	7-4	Segment Header	7-4
Execute	7-5	Execute	7-5	Execute	7-6	Execute	7-6
				Transfer	7-6	Transfer	7-6
Origin	7-7	Origin	7-8	Origin	7-8	Origin	7-8
Modular Origin	7-9	Modular Origin	7-9	Modular Origin	7-9	Modular Origin	7-9
		Literal Origin	7-10	Literal Origin	7-10	Literal Origin	7-10
Admode	7-11	Admode	7-11	Admode	7-12	Admode	7-12
Equals	7-12	Equals	7-12	Equals	7-13	Equals	7-13
Control Equals	7-13	Control Equals	7-13	Control Equals	7-14	Control Equals	7-14
Memory Dump	7-14						

Table 7-1 (cont). Assembly Control Statements

Easycoder A		Easycoder B		Easycoder C		Easycoder D	
Assembly Control Statements	Page Ref.	Assembly Control Statements	Page Ref.	Assembly Control Statements	Page Ref.	Assembly Control Statements	Page Ref.
				Skip	7-15	Skip	7-15
				Suffix	7-15	Suffix	7-15
				Repeat	7-16	Repeat	7-16
				Generate	7-17	Generate	7-17
				Set Line Number	7-18	Set Line Number	7-18
				Set Out-of-Sequence Base	7-18	Set Out-of-Sequence Base	7-18
Clear	7-19	Clear	7-20	Clear	7-20	Clear	7-20
End	7-21	End	7-21	End	7-22	End	7-22

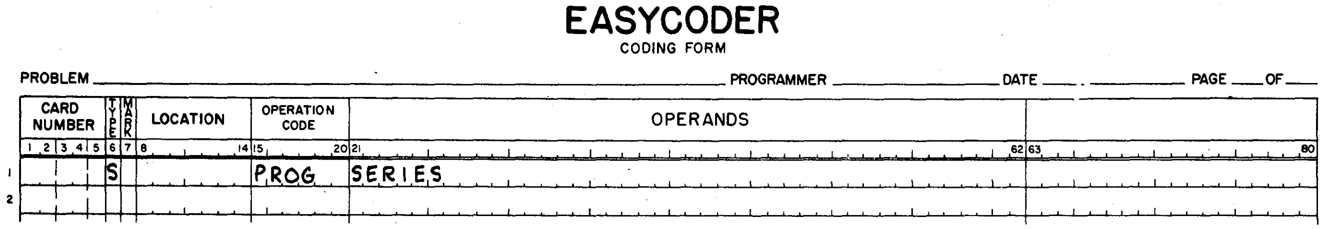
Program Header
PROG

A	B	C	D
▨	▨	▨	▨

The program header must be the first entry in a symbolic program. This statement is coded as follows for the various assembly systems.

EASYCODER A

The letters PROG must be written in the op code field, and the operands field must contain a name which identifies the program. (This name will appear in the program listing.) Optionally, an "S" can be placed in column 6; this action specifies that a check is to be made on the card number sequence of the input deck.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1		S			PROG	SERIES	
2							

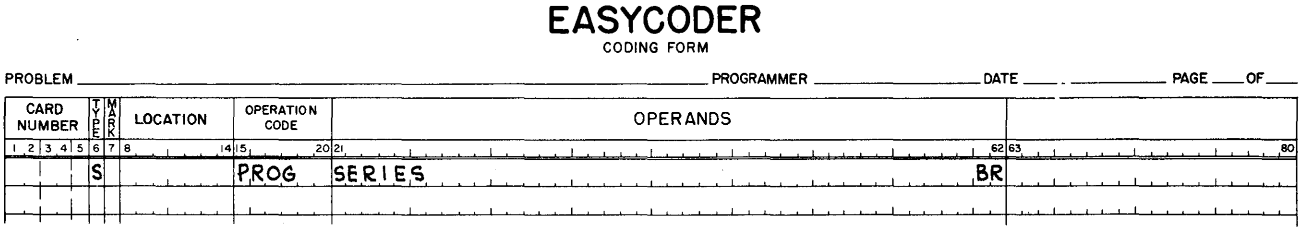
In the sample statement above, SERIES is specified as the program name, while the letter S in column 6 designates that a sequence check is desired.

EASYCODER B

The letters PROG must be written in the op code field, and the operands field must contain a name which identifies the program. (This name will appear in the program listing.) Optionally, an "S" can be placed in column 6; this action specifies that a check is to be made on the card number sequence of the input deck.

In addition, the desired object-program format is specified by the entries in columns 61 and 62. Blanks in these two columns specify that the machine-language output is to appear in the condensed-card self-loading format. Placing the letters BR in these columns specifies that the machine-language program is to appear on punched cards in BRT format. (See Easycoder B Assembly System, Order No. 011.)

NOTE: When BRT format is specified, a segment number of 01 is generated by the assembler for the first segment (memory load) following the program header. If Execute statements (see page 7-5) appear in the symbolic program, subsequent segment names are ,generated by incrementing the previous segment number by one.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ____ . ____ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1		S			PROG	SERIES BR	
2							

The statement above designates SERIES as the program name and specifies that a sequence check is to be performed. As columns 61 and 62 contain the letters BR, the output will appear on punched cards in BRT format.

EASYCODER C

As used in Easycoder C, the program header provides program identification; in addition, however, this statement serves as the all-important "action director" statement. For this reason, the programmer should refer to the Honeywell publication Easycoder Assemblers C and D, Order No. 041 for a detailed description.

EASYCODER D

As used in Easycoder D, the program header provides program identification; in addition, however, this statement serves as the all-important "action director" statement. For this reason, the programmer should refer to the Honeywell publication Easycoder Assemblers C and D, Order No. 041.

If the programmer desires to use the alternate card format (which allows room for tags consisting of up to ten characters, see page 5-10), column 75 of the program header card must contain the letter A. The PROG card itself, however, is never coded in the alternate format: the letters PROG always appear in the op code field (columns 15 through 18), while the name of the program always appears beginning in column 21.

NOTE: If the alternate format is specified, all cards following the program header, up to and including the END card, must be coded in the alternate format.

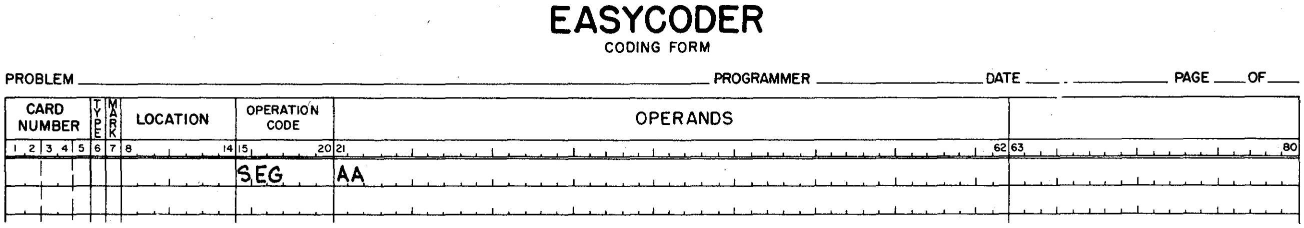
Segment Header
SEG

A	B	C	D
		▨	▨

Programs written for Easycoder C or D may be divided into two or more segments, each of which is loaded into memory and executed as a unit. It is the function of the SEG statement to define the beginning of each segment (memory load). Use of the SEG statement is optional, however. If used, a SEG statement must follow the program header, each Execute statement and each Transfer statement. If it is desired to omit this statement, it must be omitted from the entire program; in this case the assembler generates segment identifications (starting with 01).

EASYCODER C AND D

The letters SEG must be placed in the op code field, while the operands field must contain a two-character segment identification. This segment identification becomes appended to the program name to form a unique search code.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
1					SEG	AA	
2							

In the example above, AA could represent the first segment of a program, in which case this entry would follow the program header.

Execute
EX

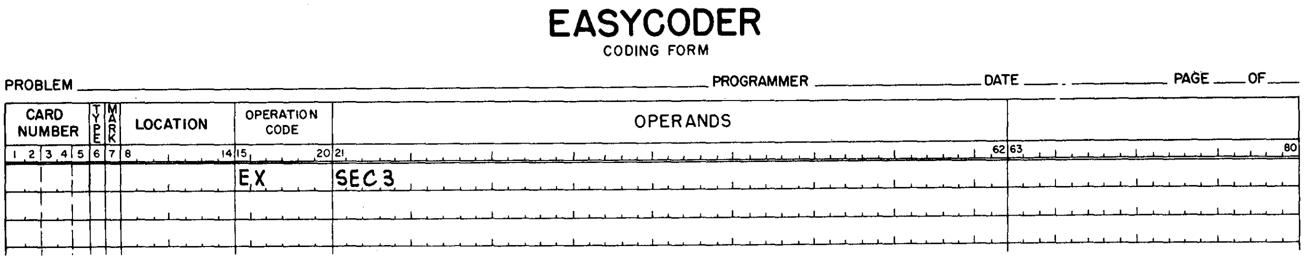
A	B	C	D
▨	▨	▨	▨

The end of a memory load is indicated by an EX statement. When the coding inserted by the assembler for the EX statement is encountered during the loading process, a branch to the location specified in the operands field results. This operation enables portions of the program to be executed before the entire program has been loaded. The coding to be executed must appear prior to the EX statement.

EASYCODER A

The letters EX must be written in the op code field; the operands field contains a direct address, either absolute or symbolic. (If an EX statement is written with a blank operands field, the machine will halt when it encounters the corresponding coding during the loading operation.)

To resume the loading operation, the last instruction in the portion of the program executed must be a Branch instruction which provides re-entry to the load routine. In addition, the first instruction of the executed routine should be an SCR (Store Control Registers) instruction which stores the contents of the B-address register in the A address of the return Branch instruction.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					EX	SEC3	
2							
3							

The sample statement above illustrates an EX statement with a symbolic address in the operands field. When the corresponding coding is encountered during the loading operation, program loading is temporarily halted and the portion of the program beginning at the location tagged SEC3 is executed.

EASYCODER B

The letters EX must be written in the op code field; the operands field contains a direct address, either absolute or symbolic. (If an EX statement is written with a blank operands field, the machine will halt when it encounters the corresponding coding during the loading operation.)

To resume the loading operation, the last instruction in the portion of the program executed must be a Branch instruction which provides re-entry to the load routine. In addition, the first instruction of the executed routine should be an SCR (Store Control Registers) instruction which stores the contents of the B-address register in the A address of the return Branch instruction.

Besides causing a branch to the programmer's coding, use of the EX statement causes any literals used in the memory load to be loaded and the literal table to be cleared. If a LITORG statement (see below) does not precede the EX statement, literals are allocated immediately following the in-line coding for the memory load.

NOTES: 1. Following an EX statement, a new segment number is generated as explained above in the description of the program header.

2. With Easycoder B, the total of the numbers of Execute, Literal Origin, and End statements must not exceed 31.

See the sample statement given above for Easycoder A.

EASYCODER C AND D

The letters EX must be written in the op code field; the operands field must contain a direct address, either absolute or symbolic. When used with these assemblers, the EX statement enables a program to be loaded and executed one segment at a time. Each segment except the last must end with either an EX or an XFR statement. When an EX statement is encountered, all literals preceding the EX statement which have not been allocated to memory are allocated in sequence, and the literal table is cleared.

Note that it is the responsibility of the programmer to provide re-entry to the load routine. The methods of returning to the applicable loader are described in the pertinent Honeywell publication - e.g., Card Loader-Monitor B (Order No. 154) or Tape Loader-Monitor C (Order No. 221).

See the sample statement given above for Easycoder A.

Transfer
XFR

A	B	C	D
		▨	▨

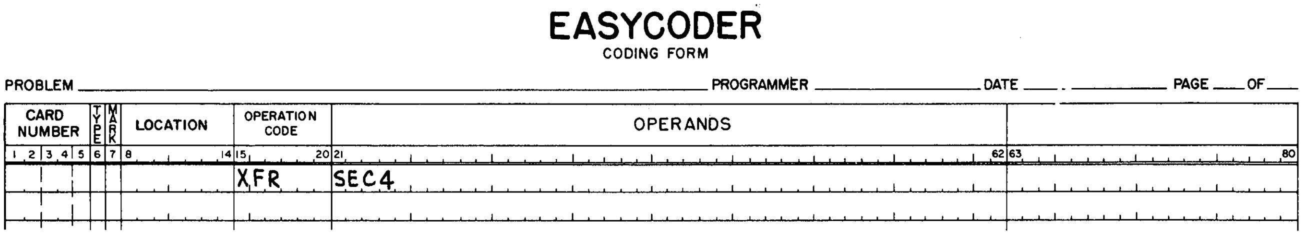
For Easycoder C and D users, the end of a memory load may be indicated by an XFR statement instead of an EX statement. Both statements perform essentially the same functions; the one exception is that use of the XFR statement does not result in the allocation of literals or in the clearing of the literal table.

When the coding inserted by the assembler for the XFR statement is encountered during the loading process, a branch to the location specified in the operands field results. This operation enables portions of the program to be executed before the entire program has been loaded.

EASYCODER C AND D

The letters XFR must be written in the op code field; the operands field must contain a direct address, either absolute or symbolic. Use of this statement enables a program to be loaded and executed one segment at a time. Each segment except the last must end with either an XFR or an EX statement.

NOTE: It is the responsibility of the programmer to provide re-entry to the load routine.



EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ____ OF ____

	CARD NUMBER 1-5	TYPE 6	MARK 7	LOCATION 8-14	OPERATION CODE 15-20	OPERANDS 21-62	63-80
1					XFR	SEC4	
2							

The sample statement above illustrates an XFR statement with a symbolic address in the operands field. When the corresponding coding is encountered during the loading operation, program loading is temporarily halted and the portion of the program beginning at the location tagged SEC4 is executed.

Origin
ORG

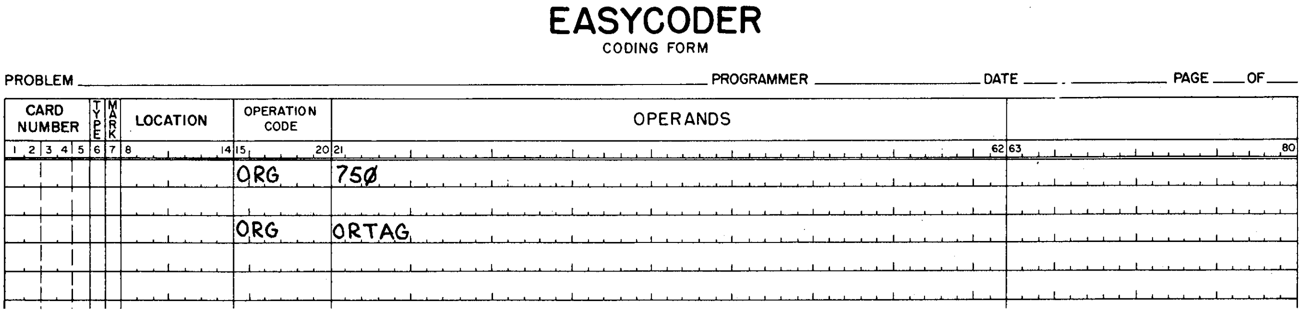
A	B	C	D
▨	▨	▨	▨

The ORG statement is used to modify the normal memory allocation process of assembly. This statement can be inserted anywhere in the source program to indicate to the assembler that all subsequent coding (instructions, constants, work areas, etc.) should be assigned sequential memory locations starting with the location whose address is specified in the operands field.

A program is normally allocated memory space beginning at location 0. If it is desired to assign memory space starting at some location other than 0, an ORG statement must be inserted in the program immediately following the program header.

EASYCODER A

The letters ORG are written in the op code field, and an address (either absolute or symbolic) is written in the operands field. (If the address is symbolic, the tag must appear in the location field of a previous source-program entry.) The address specified in the operands field is assigned the tag (if any) in the location field; if this tag appears, it must not be indented.



EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ____ OF ____

	CARD NUMBER 1-5	TYPE 6	MARK 7	LOCATION 8-14	OPERATION CODE 15-20	OPERANDS 21-62	63-80
1					ORG	75Ø	
2							
3					ORG	ORTAG	
4							
5							

The first statement above indicates to the assembler that all subsequent entries should be assigned sequential addresses beginning with location 750. The second statement directs the assembler to assign to all subsequent entries sequential addresses beginning with the address that is assigned to the tag ORTAG. (ORTAG must appear in the location field of a previous source-program entry.)

EASYCODER B

The letters ORG are written in the op code field, and an address (either absolute or symbolic) is written in the operands field. (If the address is symbolic, the tag must appear in the location field of a previous source-program entry.) The address specified in the operands field is assigned the tag (if any) in the location field; if this tag appears, it must not be indented.

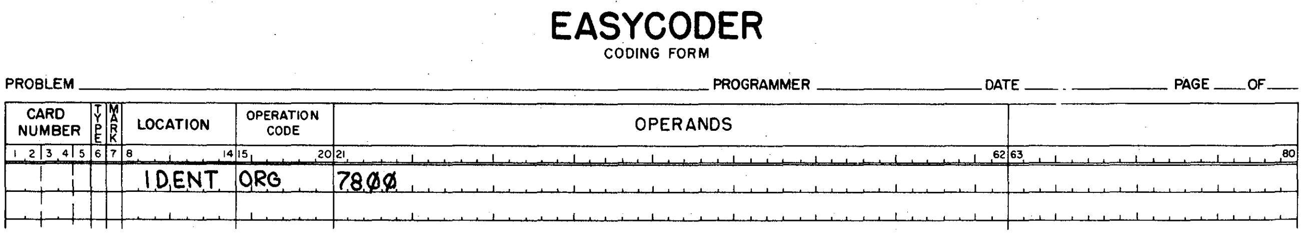
NOTE: When the BRT punched-card format is specified, an ORG statement must be included immediately following the PROG statement with an address of 1,000 (decimal) or above.

See the sample statements given above for Easycoder A.

EASYCODER C AND D

The letters ORG are written in the op code field, and an address (either absolute or symbolic) is written in the operands field. If the address is symbolic, the tag must appear in the location field of another (not necessarily previous) source-program entry. A symbolic tag may be written in the location field. If this tag begins in column 8, it is assigned to the address written in the operands field. If it begins in column 9, the tag is assigned to the location at which the next instruction would have begun had the ORG statement not been present.

NOTE: Care must be taken so that the address in the operands field is a decimal number of 1,000 or above if Card Loader-Monitor B is used to load the object program. If Tape Loader-Monitor C or Drum Bootstrap-Loader C is used, this decimal number must be 1,340 or above.



EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ____ OF ____

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				IDENT	ORG	78ØØ	
2							

In the example above, assume that the instruction preceding the ORG statement was assigned to locations 5000 through 5007. The next instruction would normally begin at location 5008. The tag IDENT, since it begins in column 9, is thus assigned to location 5008, and the next instruction is stored beginning at location 7800.

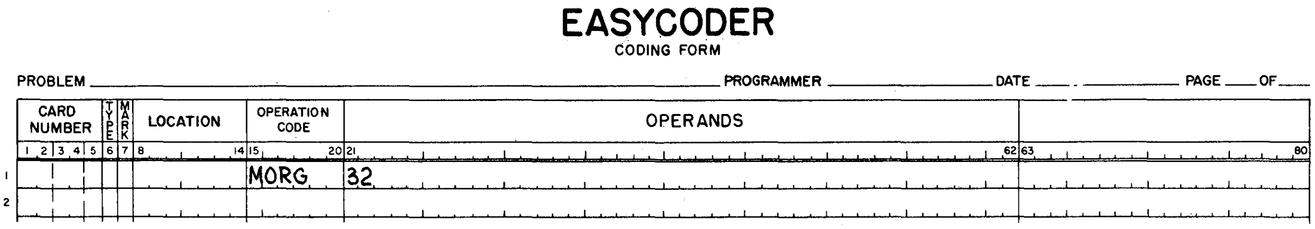
Modular Origin MORG

A	B	C	D
▨	▨	▨	▨

The modular origin statement is similar to the ORG statement described above. The MORG statement indicates to the assembler that all subsequent entries should be assigned sequential addresses starting with the next available location whose address is a multiple of the number written in the operands field of the MORG statement. The entry in the operands field must represent a power of two (e.g., 2, 4, 8, 16, 32, ... 4,096, etc.).

EASYCODER A AND B

The letters MORG are written in the op code field, and a number (a power of two) is placed in the operands field.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					MORG	32	
2							

The statement above indicates to the assembler that all subsequent entries should be assigned sequential addresses beginning with the next available location whose address is a multiple of 32.

EASYCODER C AND D

The letters MORG are written in the op code field, and a number (a power of two) is placed in the operands field. A symbolic tag may be written in the location field. If this tag begins in column 8, it is assigned to the address written in the operands field. If it begins in column 9, the tag is assigned to the location at which the next instruction would have begun had the MORG statement not been present (see the sample statement given above for the ORG statement).

Literal Origin LITORG

A	B	C	D
	▨	▨	▨

The literal origin statement is similar to the ORG and MORG statements described above. The LITORG statement specifies to the assembler that all previously used literals should be assigned sequential memory locations starting with the location specified in the operands field.

Care must be taken to ensure that literals can be referenced by the instructions which use them; e.g., a literal stored in one 4K bank may not be addressed in the two-character mode from another bank.

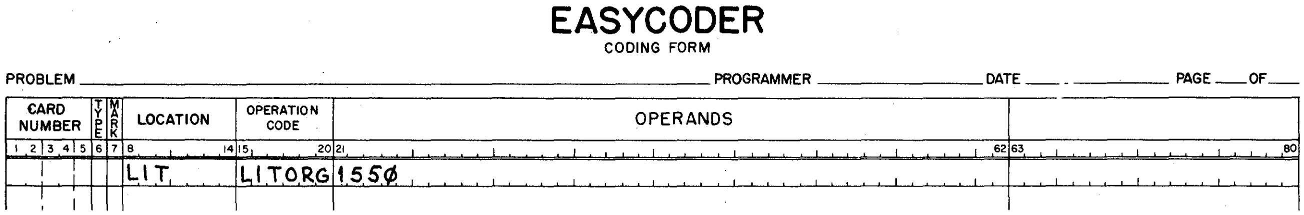
EASYCODER B

The op code field must contain the letters LITORG, while the operands field contains an address (either absolute or symbolic). If a symbolic tag is used, it must have appeared in the location field of a previous entry. Like the EX statement, the LITORG statement causes the literal table to be cleared. Also, locations below 1,000 (decimal) must not be used when BRT punched-card output is specified in the PROG statement.

A symbolic tag may be written in the location field. If this tag begins in column 8, it is assigned to the address written in the operands field. If it begins in column 9, the tag is assigned to the location at which the next instruction would have begun had the LITORG statement not been present.

NOTES: 1. In the absence of a LITORG statement, all of the generated coding associated with a memory load is allocated immediately following the in-line coding.

2. With Easycoder B, the total of the number of Execute, Literal Origin, and End statements must not exceed 31.



EASYCODER

CODING FORM

PROBLEM ________ PROGRAMMER ________ DATE ________ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				LIT	LITORG	155Ø	
2							

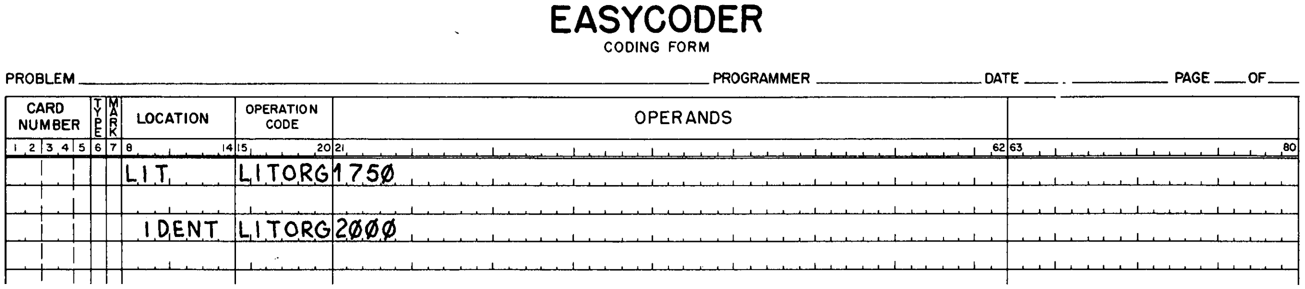
In the LITORG statement above, the assembler is directed to assign sequential addresses — starting with location 1550 — to all previously encountered literals. This location is also tagged LIT, since the tag begins in column 8.

EASYCODER C AND D

The op code field must contain the letters LITORG, while the operands field contains an address (either absolute or symbolic). If a symbolic tag is used, it must have appeared in the location field of another, not necessarily previous, entry. Like the EX statement, the LITORG statement causes the literal table to be cleared. Also, locations below 1,340 (decimal) must not be used.

A symbolic tag may be written in the location field. If this tag begins in column 8, it is assigned to the address written in the operands field. If it begins in column 9, the tag is assigned to the location at which the next instruction would have begun had the LITORG statement not been present.

NOTE: In the absence of a LITORG statement, all of the generated coding associated with a memory load — except for a memory load terminated by an XFR statement — is allocated immediately following the in-line coding.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
1				LIT	LITORG	175Ø	
2							
3				IDENT	LITORG	2ØØØ	
4							

In the first LITORG statement above, the assembler is directed to assign sequential addresses, starting with location 1750, to all previously encountered literals. Note that the tag for this statement, LIT, begins in column 8. Assume, in the second statement above, that the instruction preceding the LITORG statement was assigned to locations 450 through 457. The next instruction would normally begin at location 458. The tag IDENT, since it begins in column 9, is thus assigned to location 458, and previously encountered literals are assigned sequential addresses starting with location 2000.

Set Address Mode
ADMODE

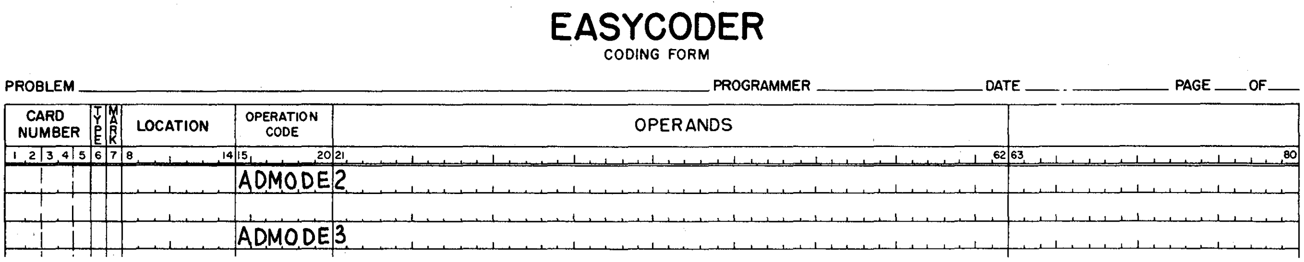
A	B	C	D
▨	▨	▨	▨

This statement specifies the addressing mode into which all subsequent instructions are to be assembled (i.e., two-, three-, or four-character). (All machine instructions, as well as the DSA data formatting statement, are affected by the address mode.) The mode of address assembly specified in this statement remains in effect until another ADMODE statement, specifying a different mode of assembly, is encountered.

Because the ADMODE statement concerns itself only with the source program, it should be used in conjunction with the CAM (Change Addressing Mode) instruction (see page 8-62). The CAM instruction specifies the addressing mode in which the machine is directed to interpret the address portions of all subsequent object-program instructions.

EASYCODER A and B

The letters ADMODE are placed in the op code field. The operands field contains either a 2 or a 3 to denote whether all subsequent instructions are to be assembled in the two-character or the three-character addressing mode. If an ADMODE statement is not included at the beginning of the source program, assembly begins in the two-character addressing mode. (It should be a general rule, however, to include an ADMODE statement at the outset of every program.)



EASYCODER
CODING FORM

PROBLEM ________ PROGRAMMER ________ DATE ________ PAGE ___ OF ___

	CARD NUMBER 1 2 3 4 5	TYPE 6	MARK 7	LOCATION 8 14	OPERATION CODE 15 20	OPERANDS 21 62	63 80
1					ADMODE	2	
2							
3					ADMODE	3	

The assembler upon encountering the first statement above will assemble the address portions of all subsequent instructions as two-character addresses. The second statement, if encountered later in the same source program, will cause the assembler to change to three-character address assembly.

EASYCODER C AND D

The letters ADMODE are placed in the op code field. The operands field contains either the numbers 2, 3, 4, or a symbolic tag to denote whether all subsequent instructions are to be assembled in the two-, three-, or four-character addressing mode. If a symbolic tag is used, it must have been previously defined to have a value of 2, 3, or 4. If an ADMODE statement is not included at the beginning of the source program, three-character addressing is assumed by the assembler. (It should be a general rule, however, to include an ADMODE statement at the outset of every program.) See the sample statements given above for Easycoder A and B.

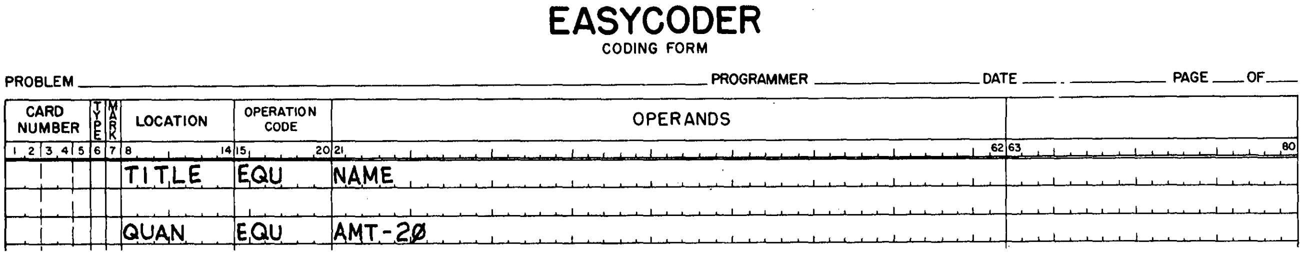
Equals
EQU

A	B	C	D
▨	▨	▨	▨

The EQU statement assigns the symbolic tag written in the location field to the address (absolute or symbolic) written in the operands field. This statement thus makes it possible to use different symbolic tags in different parts of the source program to refer to the same memory location.

EASYCODER A and B

The location field contains a symbolic tag, while the op code field contains the letters EQU. The operands field contains the address to which the symbolic tag in the location field is to be assigned. (Each symbolic tag written in the operands field must appear in the location field of a previous source-program entry.)



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				TITLE	EQU	NAME	
2							
3				QUAN	EQU	AMT-2Ø	

The first EQU statement above causes the assembler to assign the tag TITLE the same location assigned the tag NAME. Thus, the programmer can use either of these two tags to refer to the contents of this location. The second statement employs relative addressing. The assembler will assign the tag QUAN to the location specified by address arithmetic as AMT-20.

EASYCODER C AND D

The location field contains a symbolic tag, while the op code field contains the letters EQU. The operands field contains the address to which the symbolic tag is to be assigned. A symbolic tag written in the operands field must appear in the location field of another (not necessarily previous) source program entry.

See the sample statement given above for Easycoder A and B.

Control Equals CEQU

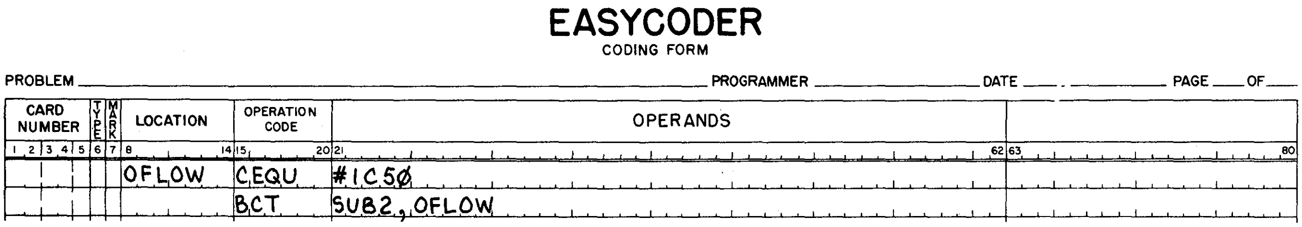
A	B	C	D
▨	▨	▨	▨

The CEQU statement assigns the symbolic tag written in the location field to the value written in the operands field. It is frequently used to assign a tag (containing a minimum of two characters) to a variant character or to a group of input/output control characters.

EASYCODER A AND B

The location field contains a symbolic tag, while the op code field contains the letters CEQU. The operands field contains an octal value; this entry is coded as an octal constant and may contain up to 12 octal digits. The symbolic tag in the location field is assigned to this entry.

NOTE: A description of octal constants may be found under the heading "Define Constant with Word Mark — DCW" (see page 6-2).



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				OFLOW	CEQU	#1C5Ø	
2					BCT	SUB2, OFLOW	

The sample coding above illustrates how a symbolic tag can be used in place of a variant character. The CEQU statement directs the assembler to equate the tag OFLOW to the octal value 50. The second line of coding contains a branch instruction which specifies that a program should branch to the location tagged SUB2 if the condition specified by the variant character tagged OFLOW is present.

EASYCODER C AND D

The location field contains a symbolic tag, while the op code field contains the letters CEQU. The entry in the operands field must be a decimal, binary, octal, or alphanumeric constant (the octal format is most commonly used). Regardless of the constant used, however, the resultant value must not exceed four characters in length.

NOTES: 1. Instructions which refer to the tag defined by the CEQU statement must not precede the CEQU statement.

2. A description of constants may be found under the heading "Define Constant with Word Mark — DCW" (see page 6-2).

See the sample statement given above for Easycoder A and B.

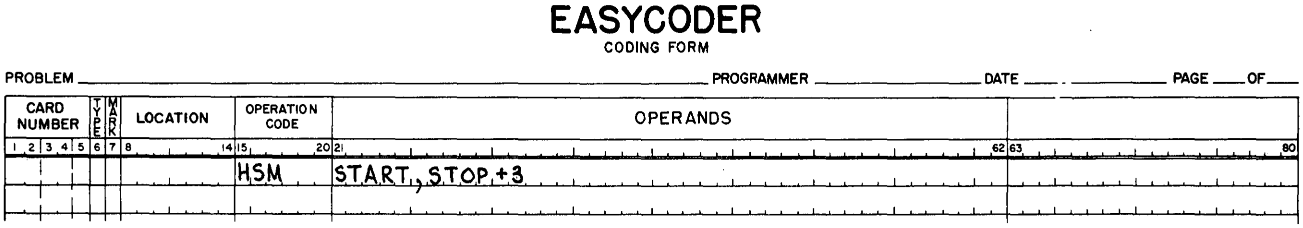
Memory Dump
HSM

A	B	C	D
▨			

The HSM statement may be used with Easycoder A to produce a punched card deck containing the Memory Dump routine. This card deck can be loaded into memory to obtain a printed listing of the contents of any portion of main memory. This statement must be coded immediately preceding the CLEAR and END statements in the source program (see below).

EASYCODER A

If the punched card deck (containing the Memory Dump routine) is to be loaded into a specific memory area, the start of this area can be specified by a tag in the location field of the HSM statement. A blank location field causes the Memory Dump routine to be loaded into the area following the location assigned to the last character in the object program. The letters HSM must be written in the op code field. The operands field contains the addresses of the first (low) and last (high) locations in the memory area whose contents are to be listed by the Memory Dump routine.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					HSM	START,STOP+3	
2							

The HSM statement above specifies that the area whose contents are to be listed begins at the location tagged START and ends three locations beyond the location tagged STOP. As the location field is blank, the Memory Dump routine will be stored in the area following the location assigned to the last character in the object program.

Skip
SKIP

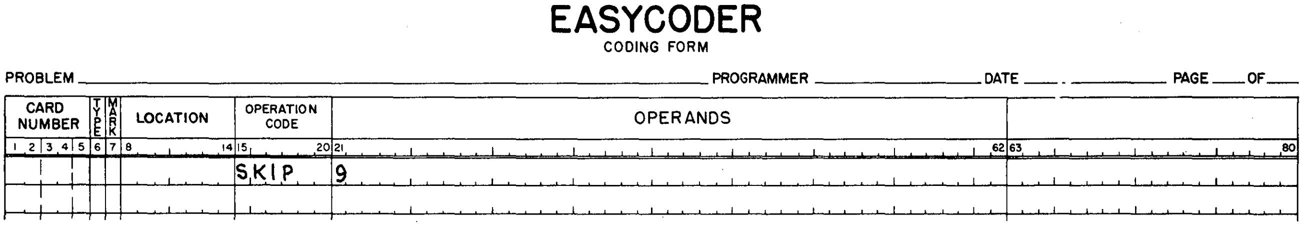
A	B	C	D
		▨	▨

Easycoder assemblers normally single-space an assembly listing and skip to the head of the next form when a page becomes filled. The SKIP statement enables the programmer to control the vertical spacing of the assembly listing by causing as many as 15 lines to be skipped.

EASYCODER C AND D

The letters SKIP are placed in the op code field. The operands field contains either a number from 1 to 15 (to indicate the total number of lines to be skipped) or the letter H (which causes the printer to skip to the head of the next form).

NOTE: The assembler automatically skips to the head of the form for each new segment.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					SKIP	9	
2							

In the sample coding above, the assembler is directed to skip 9 lines on the program listing.

Suffix
SFX

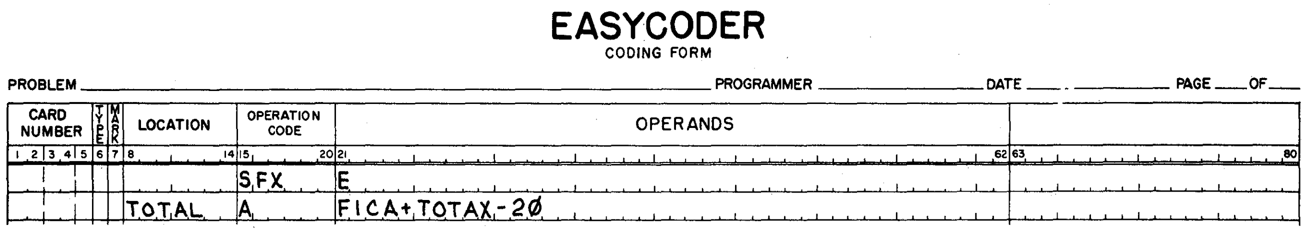
A	B	C	D
		▨	▨

The SFX statement directs the assembler to append the single-character suffix in the operands field to each tag of five characters or less contained in the following coding. This technique enables the programmer to assign unique tags for each segment of a program and thus guard against double definition of a tag between distinct segments of a program. When inter-segment referencing within a program is required, six-character tags may be assigned.

This operation continues until the occurrence of another SFX statement with a blank operands field, or until the END statement is encountered.

EASYCODER C AND D

The letters SFX are placed in the op code field. A single-character suffix is written in the operands field.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ____ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					SFX	E	
2				TOTAL	A	FICA+TOTAX-2Ø	

In the above example, the assembler interprets the Add instruction following the SFX statement as: TOTALE A FICAE+TOTAXE-20.

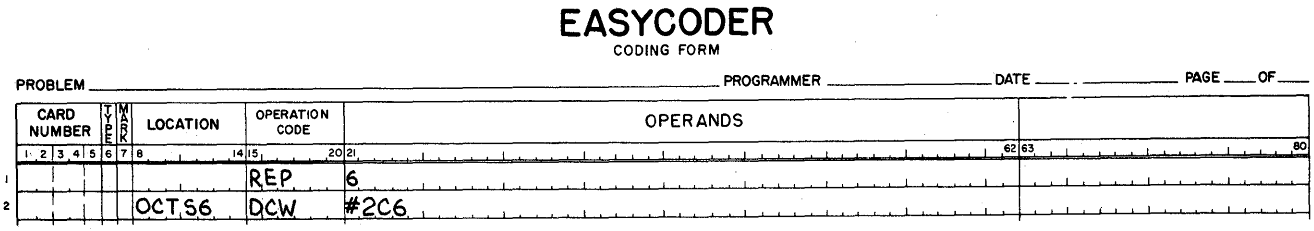
Repeat
REP

A	B	C	D
		▨	▨

This statement directs the assembler to repeat the following data formatting statement the number of times specified in the operands field. The number of times a statement is repeated includes the original statement and may not exceed 63. The assembler repeats the statement without variation, except that any entry in the location field is not repeated.

EASYCODER C AND D

The letters REP are written in the op code field. The operands field designates the number of times the following statement is to be repeated (including the original statement).



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ____ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					REP	6	
2				OCTS6	DCW	#2C6	

In the sample statement above, REP is employed to define six identical constants of octal value 6000.

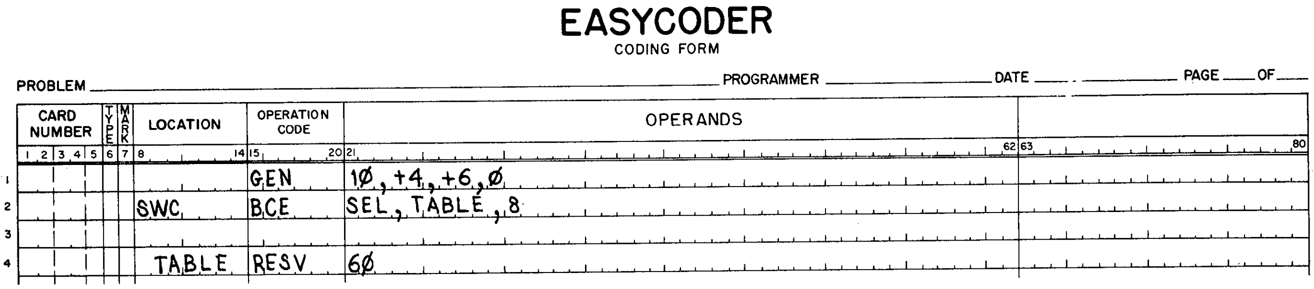
Generate GEN

A	B	C	D
		▨	▨

This statement directs the assembler to generate the instruction which follows a specified number of times, incrementing or decrementing the operands of the instruction as specified by the operands field of the GEN statement. The GEN statement can apply to machine instructions with formats containing a single address, both addresses, a single address and one variant character, or both addresses and one variant character (only one variant character is allowed).

EASYCODER C AND D

The letters GEN are written in the op code field. The operands field contains the parameter specifying the number of times the statement (which follows) is to be generated, including the original statement. This number is followed by a modifier for each operand in the model statement. These modifiers specify the increment (from 0 to +63) or decrement (from -63 to 0) to be applied to each of the operands each time the statement is generated. There must be a modifier for each operand in the model statement (including the variant character, if any), and the modifiers must appear in the same order as the operands. If no modification is desired, 0 is entered as the modifier.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					GEN	1Ø,+4,+6,Ø	
2				SWC	BCE	SEL,TABLE,8	
3							
4				TABLE	RESV	6Ø	

In the example above, the GEN statement generates a series of 10 instructions that will branch to a location SEL, SEL+4, SEL+8, ... or SEL+36, provided that an 8 is present in the first character of the corresponding item in a table containing 10 six-character items. The tag SWC is assigned to the leftmost character of the first generated instruction. The GEN statement itself must not be tagged.

NOTE: The second BCE instruction generated by the example is BCE/SEL+4, TABLE+6, 8; the third instruction generated is BCE/SEL+8, TABLE+12, 8; and so on. The tenth instruction generated is BCE/SEL+36, TABLE+54, 8.

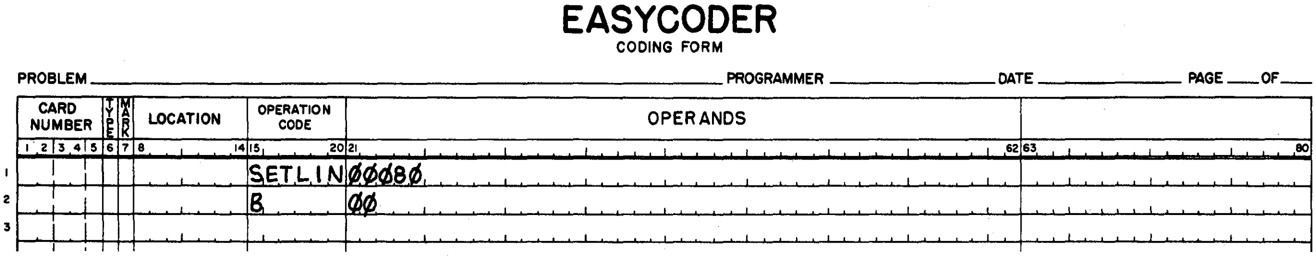
Set Line Number
SETLIN

A	B	C	D
		▨	▨

This instruction is used to control the generation of line numbers by the assembler.

EASYCODER C AND D

The letters SETLIN are written in the op code field, while the first five columns of the operand field contain the desired line number. The assembler replaces the contents of the line number generation counter with the number in the operands field of the SETLIN statement. This statement is effective only when the assembler is generating line numbers. It is important to note that all of the first five columns in the operands field must be punched with a decimal number (i.e., leading zeros are required).



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER 1 2 3 4 5	TYPE 6	MARK 7	LOCATION 8 14	OPERATION CODE 15 20	OPERANDS 21 62	63 80
1					SETLIN	ØØØ8Ø	
2					B	ØØ	
3							

In the example above, the SETLIN statement causes the instruction which follows it (B/00) to be assigned a line number of 00080.

Set Out-of-Sequence Base
XBASE

A	B	C	D
		▨	▨

The XBASE statement establishes the out-of-sequence base (OSB). As its name implies, the OSB is a base address for the storage of out-of-sequence coding. Such coding may be allocated or referred to (1) by means of the address code ' (apostrophe) in the location field (see page 5-9); or (2) by means of the address code ' (apostrophe) in the operands field (see page 5-15).

EASYCODER C AND D

The letters XBASE are written in the op code field. The operands field contains the value (absolute or symbolic) to which the assembler is directed to set the out-of-sequence base (OSB). If a symbolic tag appears in the operands field it must have appeared in the location field of a previous source-program entry.

EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER 1 2 3 4 5	TYPE 6	MARK 7	LOCATION 8 14	OPERATION CODE 15 20	OPERANDS 21 62	63 80
1					XBASE	5ØØ	
2				'275	DCW	@CON@	
3							

In the above example, the out-of-sequence base (OSB) is set to 500 by the XBASE statement. When the second entry is encountered, the assembler assigns the rightmost character of the constant CON to location 775 (500 + 275).

Clear
CLEAR

A	B	C	D
▨	▨	▨	▨

The CLEAR statement enables the programmer to specify an area of memory which is to be cleared of punctuation before the object program is loaded. The memory area is also cleared to zeros or to a given character. It is not necessary to clear areas which will be used to store the object program.

EASYCODER A

The op code field contains the letters CLEAR, while the operands field contains the addresses (either absolute or symbolic) of the first (low) and last (high) locations in an area to be cleared. If a comma is written immediately following the second address, the character written in the column after the comma is loaded into all locations in the cleared area. If two addresses are written in the operands field and are not followed by a comma and a character, the specified area is cleared to zeros.

A number of CLEAR statements may be written (in sequence) immediately preceding the END statement, provided that the total number of HSM, CLEAR, and END statements does not exceed 10.

NOTE: The 80-character loading area specified in the END statement must never be cleared.

EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER 1 2 3 4 5	TYPE 6	MARK 7	LOCATION 8 14	OPERATION CODE 15 20	OPERANDS 21 62	63 80
1					CLEAR	CAMT,EAMT	
2							
3					CLEAR	334,379,J	

The first CLEAR statement above specifies that the area beginning at the location tagged CAMT and ending at the location tagged EAMT is to be cleared to zeros. The second CLEAR statement clears the area beginning at location 334 and ending at 379 to 46 J's.

EASYCODER B

The op code field contains the letters CLEAR, while the operands field contains the addresses (either absolute or symbolic) of the first (low) and last (high) locations in an area to be cleared. If a comma is written immediately following the second address, the character written in the column after the comma is loaded into all locations in the cleared area. If two addresses are written in the operands field and are not followed by a comma and a character, the specified area is cleared to zeros.

A maximum of nine CLEAR statements may be included in a program. In addition, no coding may appear between the last symbolic CLEAR statement and the END statement.

NOTE: The loading area specified in the END statement must never be cleared.

See the sample statements given above for Easycoder A.

EASYCODER C AND D

The op code field contains the letters CLEAR, while the operands field contains the addresses (either absolute or symbolic) of the first (low) and last (high) locations in an area to be cleared. If a comma is written immediately following the second address, the character written in the column after the comma is loaded into all locations in the cleared area. If two addresses are written in the operands field and are not followed by a comma and a character, the specified area is cleared to zeros. As many CLEAR statements as necessary can be included in a program.

NOTE: The programmer must exercise caution in the physical placement of the CLEAR statement, as the clearing is performed by the Loader at the time the CLEAR statement is encountered.

See the sample statements given above for Easycoder A.

End
END

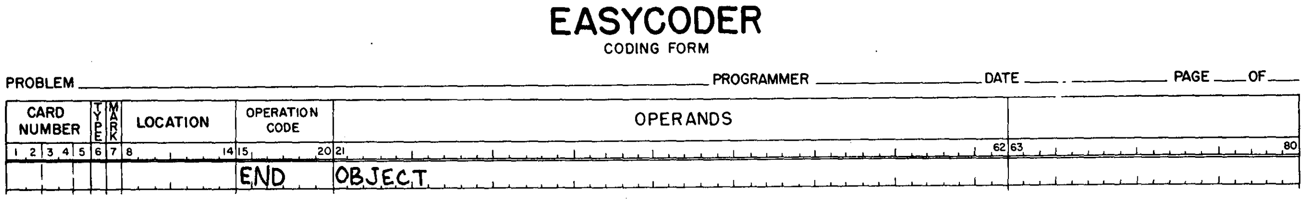
A	B	C	D
▨	▨	▨	▨

The last source program instruction must be the END statement, which indicates to the assembler that the end of the source program has been reached.

EASYCODER A

The location field may contain an address (either absolute or symbolic) which specifies the initial location in an 80-character loading area. If the location field is left blank, the assembler automatically reserves an 80-character loading area following the location assigned to the last character in the object program.

The op code field contains the letters END. If it is desired to execute the object program immediately after loading, the operands field must contain the address (either absolute or symbolic) at which the object program is to begin.



EASYCODER
CODING FORM

PROBLEM ________ PROGRAMMER ________ DATE ________ PAGE ___ OF ___

CARD NUMBER 1 2 3 4 5	TYPE 6	MARK 7	LOCATION 8 14	OPERATION CODE 15 20	OPERANDS 21 62	63 80
1				END	OBJECT	

The END statement above specifies that the object program (beginning at the address tagged OBJECT) is to be executed immediately after loading. Since the location field is blank, the assembler will reserve an 80-character loading area following the location assigned to the last character in the object program.

EASYCODER B

The method of coding this statement depends on which output format has been specified in the program header statement.

1. Output in self-loading format: The location field may contain an address (either absolute or symbolic) which specifies the initial location in an 80-character loading area. If the location field is left blank, the assembler automatically assigns an 80-character loading area following the location assigned to the last character in the object program.

   The op code field contains the letters END, while the operands field contains the address (either absolute or symbolic) to which the Loader branches when loading has been completed.

   NOTES: 1. The programmer should ensure that the loading area does not span two 4K memory banks.

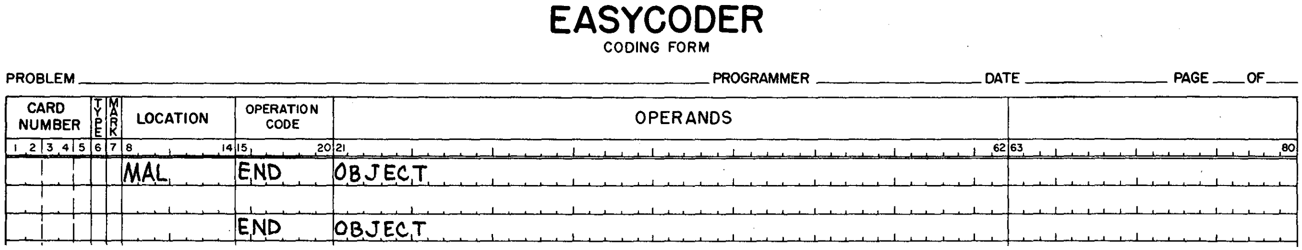
   2. During the loading process, the object program must not use the loading area. However, the area may be used following program loading.

   3. When literals are used, the programmer must specify a loading area that does not coincide with the memory area occupied by literals.

2. Output in BRT format: The op code field contains the letters END, while the operands field contains the address (either absolute or symbolic) to which the Loader branches when loading has been completed. When BRT format is specified, all other fields of the END instruction are ignored by the assembler.

NOTES: 1. The loading area is automatically assigned by the Loader.

2. With Easycoder B, the total of the numbers of Execute, Literal Origin, and End statements must not exceed 31.



EASYCODER

CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

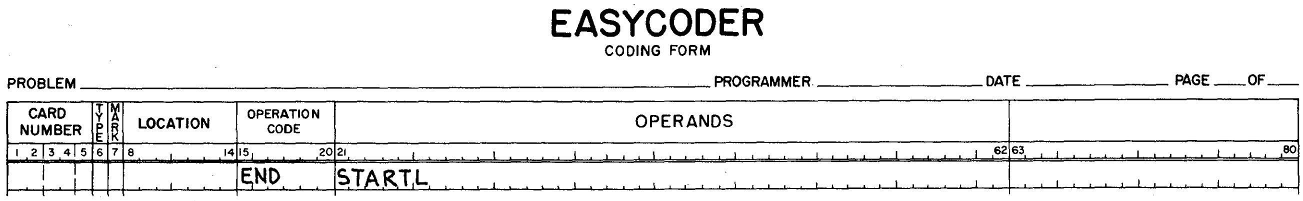
	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				MAL	END	OBJECT	
2							
3					END	OBJECT	

The first example above illustrates the coding which may be used for self-loading format output; the coding for BRT-format output is shown in the second example.

EASYCODER C AND D

The op code field contains the letters END. An address must appear in the operands field; the Loader will branch to that address (which should be the starting location of the last segment of the program).

NOTE: The loading area is automatically assigned by the Loader.



EASYCODER

CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					END	STARTL	

The sample END statement above indicates to the assembler that the end of the source program has been reached. This statement is replaced by coding which specifies to the Loader that the last (or only) segment begins at symbolic address STARTL.

8
INSTRUCTIONS

INTRODUCTION

A Series 200 computer operates under the direction of instructions in the stored program. For descriptive purposes, these instructions are classified into six functional categories: (1) Arithmetic; (2) Logic; (3) Control; (4) Interrupt Control; (5) Editing; and (6) Input/Output.

All instructions are described in the following standard format:

Title: The title describes the instruction. It appears in the left-hand margin of a page, along with the mnemonic operation code used in the Easycoder symbolic programming language.

If an instruction is included in an optional feature, that feature number accompanies the title.

Format: This is a tabular representation of all formats which may be used when coding the instruction.

Function: The function of the instruction is described in terms of each format in which it can be coded.

Word Marks: The effect of word marks with regard to data fields is specified.

Address Registers after Operation: The contents of the address registers are indicated for each of the instruction's formats.

Notes: This is additional information pertaining to the operation.

Examples: Practical applications of the instruction in its various formats are described and illustrated as symbolic program entries.

Formulas for calculating instruction execution times are presented in Appendix C.

Table 8-1 lists the abbreviations and symbols used in the description of the instructions. Those symbols used only with specific instructions are preceded by the title of the instruction to which they pertain.

Table 8-1. Symbology Used in Series 200 Instruction Descriptions

SYMBOL	MEANING
A	A address of the instruction
B	B address of the instruction
N_i	Number of characters in the instruction
N_a	Number of characters in the A field
N_b	Number of characters in the B field
N_w	Number of characters in the A or B field, whichever is smaller
NXT	Address of next sequential instruction
JI	Address of next instruction if a branch occurs
A_p	The previous setting of the A-address register (AAR)
B_p	The previous setting of the B-address register (BAR)
Divide	
N_{dd}	Number of digits in the dividend
Move and Translate	
N_{ct}	Number of characters translated
Move Item and Translate	
N_{ut}	Number of information units translated
CSR_p	Previous contents of the change sequence register (CSR)
NA_u	Number of six-bit character locations occupied by each A-item information unit (1 or 2)
NB_u	Number of six-bit character locations occupied by each B-item information unit (1 or 2)
Load Control Registers	
(A)	Contents of the field specified by the A address.
Table Lookup	
L_{ta}	The location in the table immediately to the left of the argument (or short field) that terminated the search.

ARITHMETIC

- ADD
- SUBTRACT
- BINARY ADD
- BINARY SUBTRACT
- ZERO AND ADD
- ZERO AND SUBTRACT
- MULTIPLY
- DIVIDE

ARITHMETIC OPERATIONS

Series 200 add operations (binary addition, decimal addition) treat the A operand as the augend and the B operand as the addend. The subtract operations (binary subtraction, decimal subtraction) treat the A operand as the subtrahend and the B operand as the minuend. The result of each operation is stored in the B field. These elements are summarized in Table 8-2, where a character enclosed in parentheses indicates the contents of that field.

Table 8-2. Series 200 Add and Subtract Operations

ADDITION	SUBTRACTION
$\begin{array}{r} (B) \\ +\ (A) \\ \hline (B) \end{array}$	$\begin{array}{r} (B) \\ -\ (A) \\ \hline (B) \end{array}$

BINARY ADDITION

The Binary Add instruction combines the corresponding bits of the augend and addend and produces a binary sum which is stored in the B field. This process can be most readily analyzed on a column-by-column basis. For any column in the addition, three variables are significant to the sum: the augend digit, the addend digit, and the carry from the next lower-order column. For any column, the result is fully expressed by a sum digit (1 or 0) and either a carry or no carry to the next higher-order column. Table 8-3 lists all the possible combinations of these variables.

Table 8-3. Binary Addition Table

PREVIOUS CARRY	0	0	0	0	1	1	1	1
AUGEND	0	1	1	0	0	1	1	0
ADDEND	0	1	0	1	0	1	0	1
SUM	0	0	1	1	1	1	0	0
CARRY	0	1	0	0	0	1	1	1

BINARY SUBTRACTION

The Binary Subtract instruction performs, in effect, twos-complement arithmetic.[1] When this instruction is executed, each six-bit character of the subtrahend is converted to its ones complement[2] and added to the corresponding character in the minuend, adding from right to left.

[1] The twos complement of a binary number is formed by subtracting the number from a field of all one bits and adding one to the low-order digit of the difference.

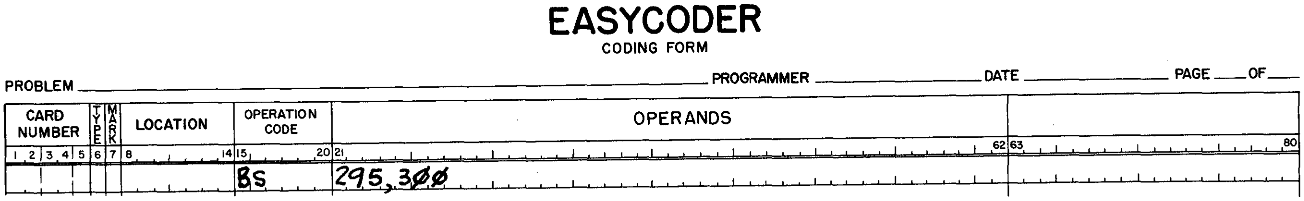
[2] The ones complement of a binary number is formed by subtracting the number from a field of all one bits.

In the first addition (the addition of the low-order characters of the subtrahend and the minuend) a simulated carry is added to the result. All subsequent characters are added with or without a carry, depending upon the result of the previous addition.

The word mark associated with the B field terminates the operation. If the length of the A field equals that of the B field, the binary subtraction process continues until the high-order B-field character has been combined with the high-order A-field character. If the length of the A field exceeds that of the B field, the effect is as if there were a word mark in the A-field location corresponding to the high-order B-field location (i.e., the process still terminates at the B-field word mark). If the length of the A field is less than that of the B field, zeros are inserted where the A field terminates until the last B-field character is processed. Each zero is converted to its ones complement as above and then added to the corresponding B-field character.

In the following example, locations 294 and 295 contain the value 73_{10} in 12-bit binary form, while locations 299 and 300 contain the binary equivalent of 87_{10}.

Note: Locations 294 and 299 contain word marks; the length of the A field therefore equals that of the B field in this example.



EASYCODER

CODING FORM

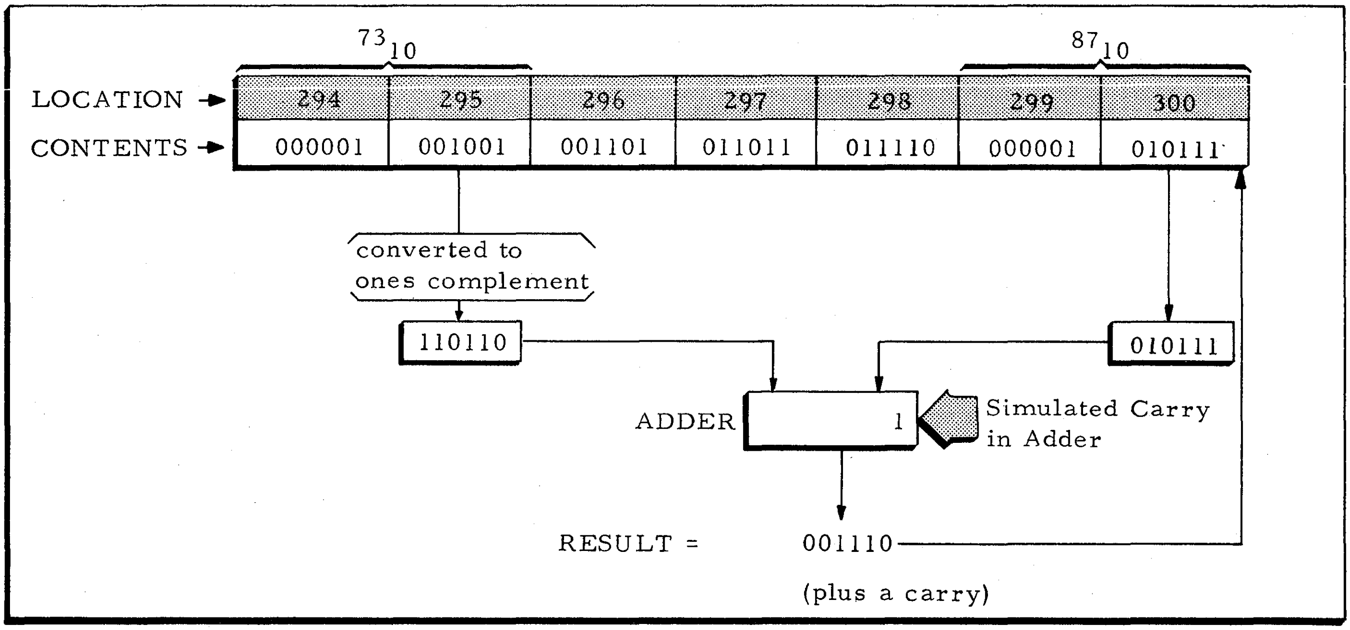
PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				BS	295,3ØØ	

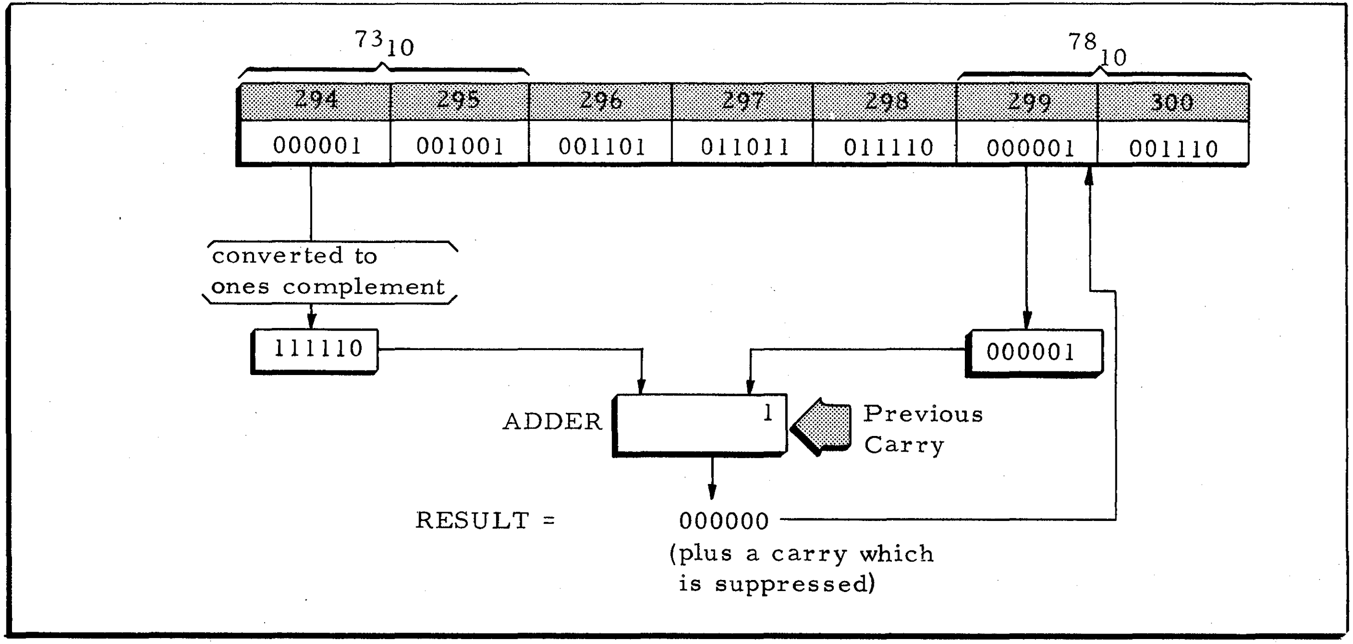
LOCATION →	294	295	296	297	298	299	300
CONTENTS → (binary)	000001	001001	001101	011011	011110	000001	010111

The six-bit character in location 295 is converted to its ones complement and added to the six-bit character in location 300 (see illustration below). Prior to this operation, a simulated carry is generated in the adder (see page 2-10). The result of the first addition is the binary equivalent of 14_{10} plus a carry. This carry remains in the adder and is added to the sum of the contents of locations 294 and 299, resulting in a binary zero plus another carry. This final carry remains in the adder and the operation terminates. An overflow condition does not exist since the carry remaining at the end of the operation is suppressed; consequently the next memory location (location 298) is not disturbed. The result of the entire Binary Subtract instruction is therefore 14_{10}, the true difference between 87 and 73.

Table 8-3 indicates how the bits in each column of the ones-complement subtrahend and the minuend are combined.



First Addition



Second Addition

The result of the operation (14_{10}) is stored in the B field as shown below.

73_{10}					14_{10}	
294	295	296	297	298	299	300
000001	001001	001101	011011	011110	000000	001110

DECIMAL ADDITION

The Add instruction performs either a true add or a complement add, depending upon the algebraic signs of the operands. The sign of an operand is determined by the combination of zone bits in the units position of that field. The four possible zone bit configurations and the signs they represent are shown in Table 8-4.

Table 8-4. Algebraic Signs in Decimal Addition

SIGN	ZONE BITS		SIGN	ZONE BITS	
	B-Bit	A-Bit		B-Bit	A-Bit
+	0	0	−	1	0
	1	1			
	0	1			

True Add

A true add is performed if the signs of the A and B fields are alike. The result of the addition is stored in the B field with the same zone bit configuration that was originally in the B field (see Figure 8-1). Zone bits in all B-field locations (except for the units position) are set to zeros. A-field zone bits (except for the units position) are ignored.

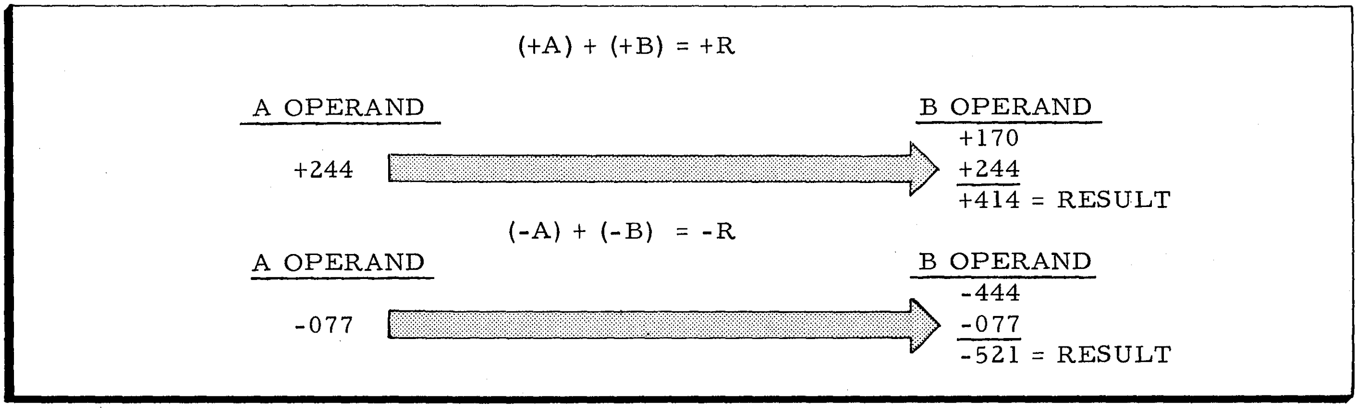


Figure 8-1. True Add Examples

Complement Add

If the operand signs are not alike, the instruction performs a complement add: the A operand is converted to its tens complement[1] and added to the B operand. The machine automatically initiates a test to determine whether a carry was generated by the high-order addition.

[1] The tens complement of a decimal number is formed by subtracting the number from all nines and adding one to the low-order digit of the difference.

The presence of a carry indicates that the result in the B field is a true answer, and the operation is terminated with the normalized sign of the B field as the sign of the result (see Figure 8-2).[1] B-field zone bits (except for the units position) are set to zeros.

The absence of a carry indicates that the A operand was algebraically larger than the B operand and that the result is stored in its tens-complement form. A recomplement cycle is performed automatically to convert the result to its true form. The sign of the result is changed during this recomplement cycle. Figure 8-2 illustrates complement add operations with and without recomplementation.

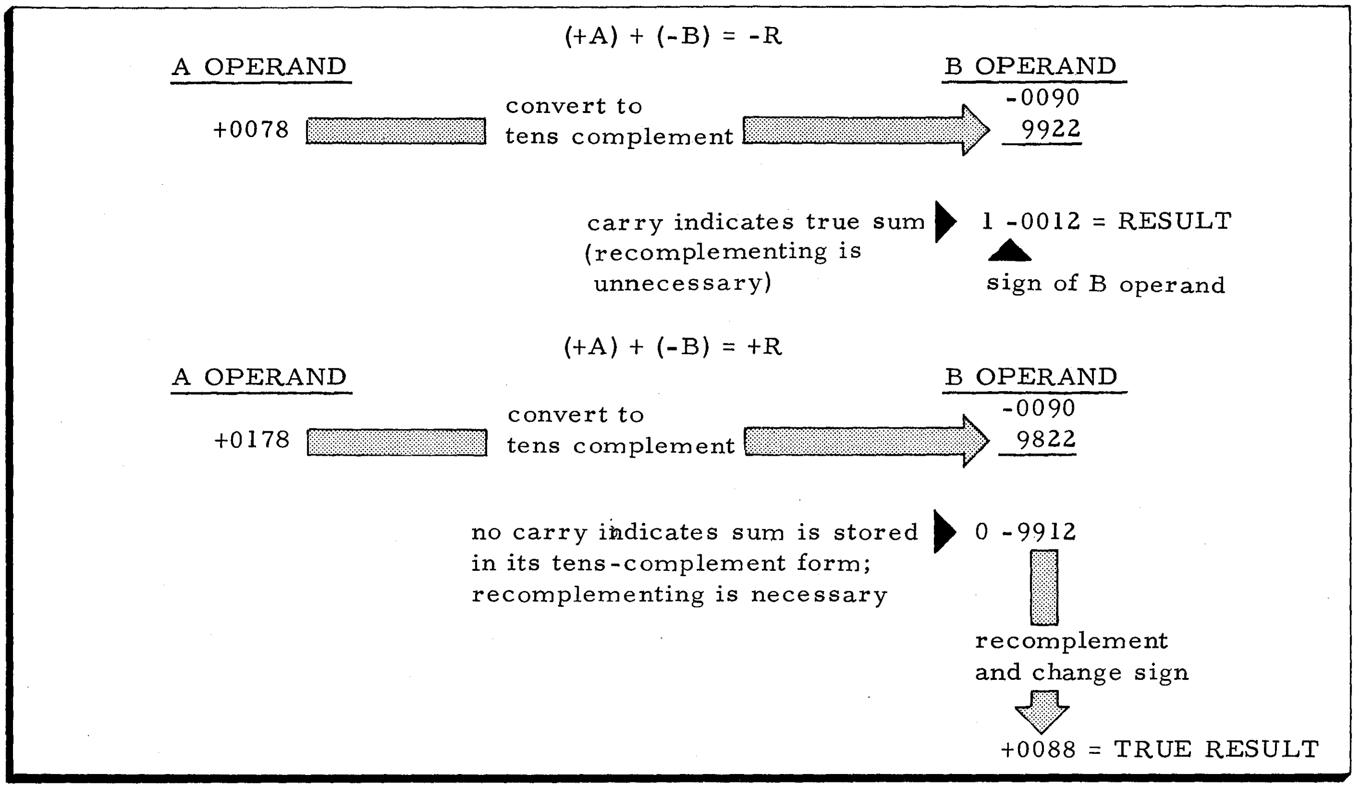


Figure 8-2. Complement Add Examples

DECIMAL SUBTRACTION

The Subtract instruction is analogous to the Add instruction with the exception that before the operands are combined, the sign of the A operand is changed. Thus, if the initial sign of the A operand is equal to that of the B operand, the operands are combined by a complement add. If, on the other hand, the initial sign of the A operand is not equal to that of the B operand, the operands are combined by a true add.

A summary of decimal arithmetic operations is presented in Table 8-5.

[1] Normalized signs are expressed by the following zone bit configurations: plus = 01, minus = 10.

Table 8-5. Decimal Arithmetic Sign Conventions

OPERATION	A-FIELD SIGN	B-FIELD SIGN	TYPE OF ADD	SIGN OF RESULT
ADD	+	+	True	+ (Bit configuration of B)
		-	Complement	Normalized sign of the operand of greater value (- = 10, + = 01)
	-	+	Complement	
		-	True	-
SUBTRACT	+	-	True	-
		+	Complement	Normalized sign of the operand of greater value (- = 10, + = 01)
	-	-	Complement	
		+	True	+ (Bit configuration of B)

INDICATORS

Two indicators are set at the completion of every decimal add and subtract operation: the overflow indicator and the zero balance indicator. If a carry is generated beyond the limit of the B field, the overflow indicator is turned on; if such a carry is not generated, the indicator is unchanged.[1] The zero balance indicator signifies either a zero or a non-zero sum. When a decimal operation produces a result equal to zero (regardless of sign), the zero balance indicator is turned on; when the result of the operation does not equal zero, the indicator is turned off.

These indicators are also set by decimal multiply and divide operations. The overflow indicator is turned on when a Decimal Divide instruction is performed in which the divisor is equal to zero. The zero balance indicator is turned on if the product of a decimal multiply operation is equal to zero.

The settings of these indicators can be tested by a Branch on Condition Test instruction (see page 8-35). This instruction automatically resets the overflow indicator; the zero balance indicator is not affected by the branch instruction used to test it but is reset only by the next decimal arithmetic instruction.

MULTIPLICATION

The Multiply instruction causes the signed decimal integer in the A field (the multiplicand)

[1] Only a "true add" operation can turn the overflow indicator on (see Table 8-5).

to be multiplied by the signed decimal integer (the multiplier) which is stored in the leftmost locations of the B field. The signed product is stored, right-justified, in the B field.

The B field must be large enough to insure an adequate number of locations for the development and storage of the product. Its length is therefore defined as the number of locations in the multiplier, plus the number of locations in the multiplicand, plus one (see Figure 8-3).

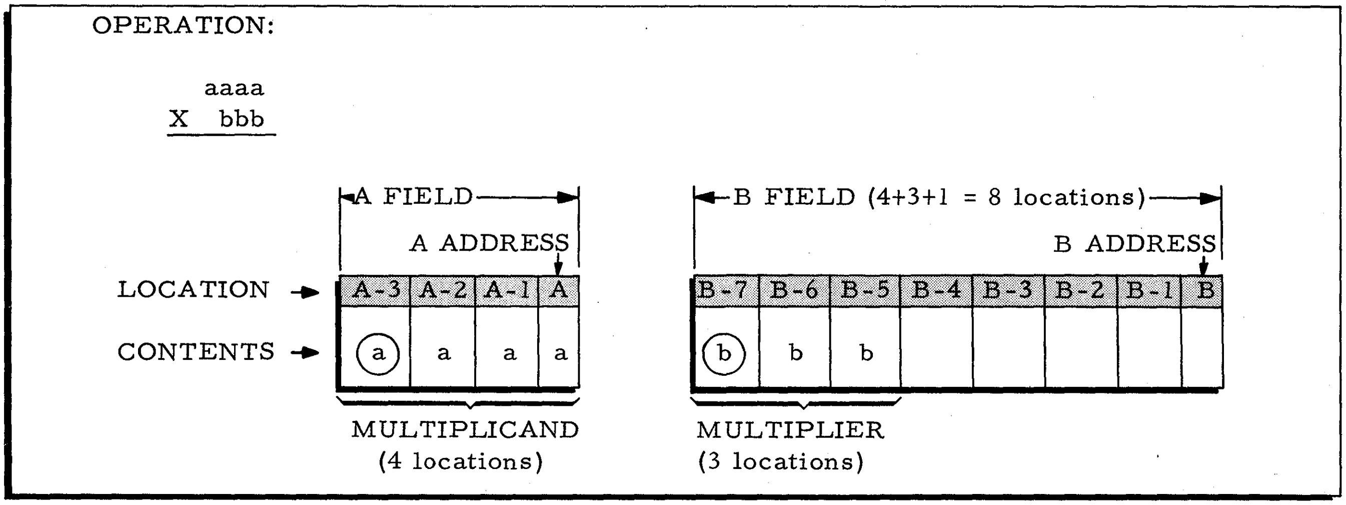


Figure 8-3. A and B Fields in Multiply Operation

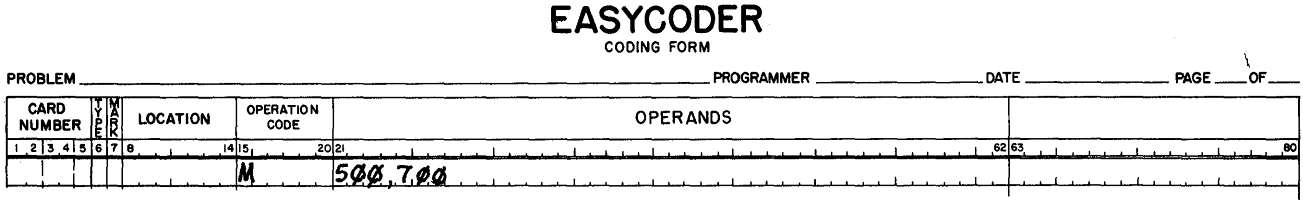
Word marks are required in the leftmost locations of the multiplicand and the multiplier. All other locations in the B field must not contain word marks. As shown in Figure 8-3, the rightmost location of the multiplier is defined as B - N_a - 1, where B is the B address and N_a is the number of locations in the A field.

The zone bits in the units positions of the multiplier and the multiplicand indicate the signs of the operands. The signs of these factors indicate the sign of the product according to the algebraic sign conventions shown in Table 8-6. The sign of the product is expressed in its normalized form (minus = 10, plus = 01).

Table 8-6. Multiply Sign Conventions

Sign of Multiplicand	+	-	+	-
Sign of Multiplier	+	-	-	+
Sign of Product	+	+	-	-

Consider the following Decimal Multiply instruction.

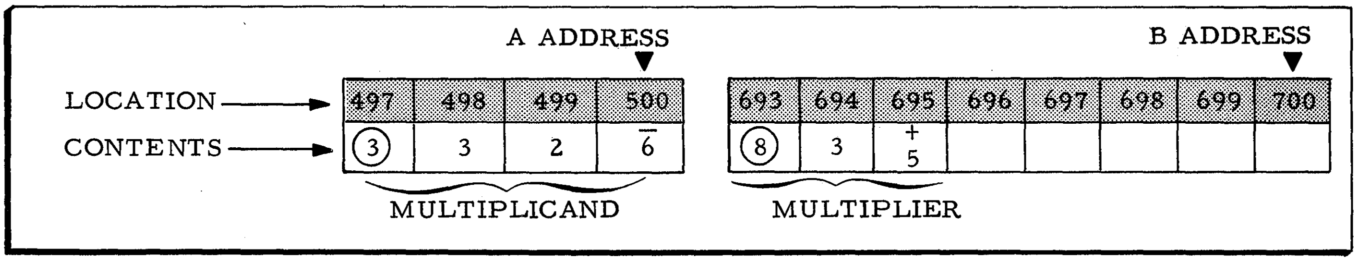


EASYCODER
CODING FORM

PROBLEM ________ PROGRAMMER ________ DATE ________ PAGE ____ OF ____

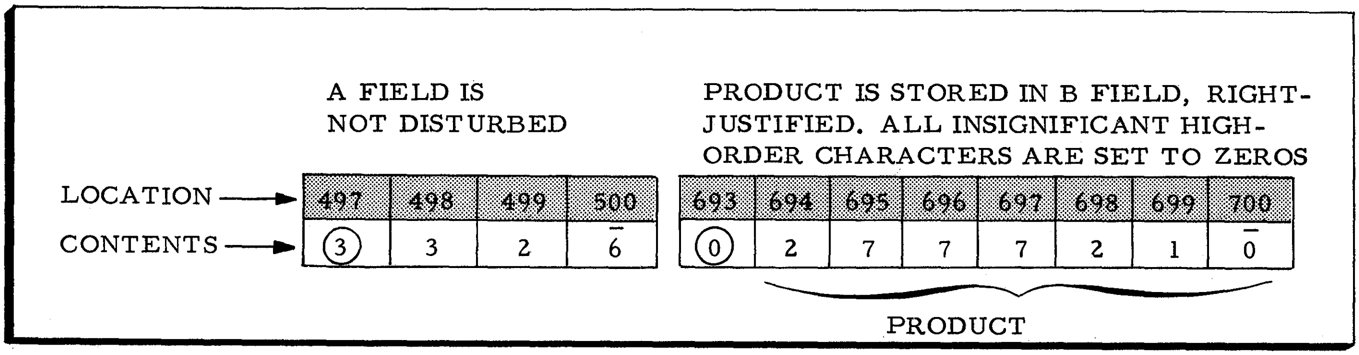
CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				M	5ØØ,7ØØ	

Location 500 is the rightmost location of a four-character field. Location 700 is the rightmost location of an eight-character field. Location 695 (i.e., 700 - 4 - 1) is the rightmost location of the multiplier.



LOCATION	497	498	499	500	693	694	695	696	697	698	699	700
CONTENTS	③	3	2	$\bar{6}$	⑧	3	$\overset{+}{5}$					

The data in the A field is multiplied by the data in the field whose rightmost location is 695, and the product is stored, right-justified, in the B field. All B-field zone bits are cleared to zeros (except in the units position, which contains the sign of the product). At the end of the operation, the multiplier is no longer present in the leftmost positions of the B field, since all B-field locations to the left of the most significant digit of the product are set to zeros. Thus, the multiplier should be preserved in another storage field if it is to be used more than once. The result of the multiply operation is shown below.


A FIELD IS NOT DISTURBED
PRODUCT IS STORED IN B FIELD, RIGHT-JUSTIFIED. ALL INSIGNIFICANT HIGH-ORDER CHARACTERS ARE SET TO ZEROS
PRODUCT

LOCATION	497	498	499	500	693	694	695	696	697	698	699	700
CONTENTS	③	3	2	$\bar{6}$	⓪	2	7	7	7	2	1	$\bar{0}$

DIVISION

The Divide instruction causes the signed decimal integer in the A field (the divisor) to be divided into the signed decimal integer whose leftmost location is the B address of the instruction (the dividend). The quotient is developed and stored in the leftmost locations of the B field, and the remainder is stored in the rightmost locations of the B field.[1] To insure an adequate number of storage locations for the development of the quotient, the length of the B field is determined by adding 1 to the sum of the number of character locations in the divisor and dividend (see Figure 8-4).

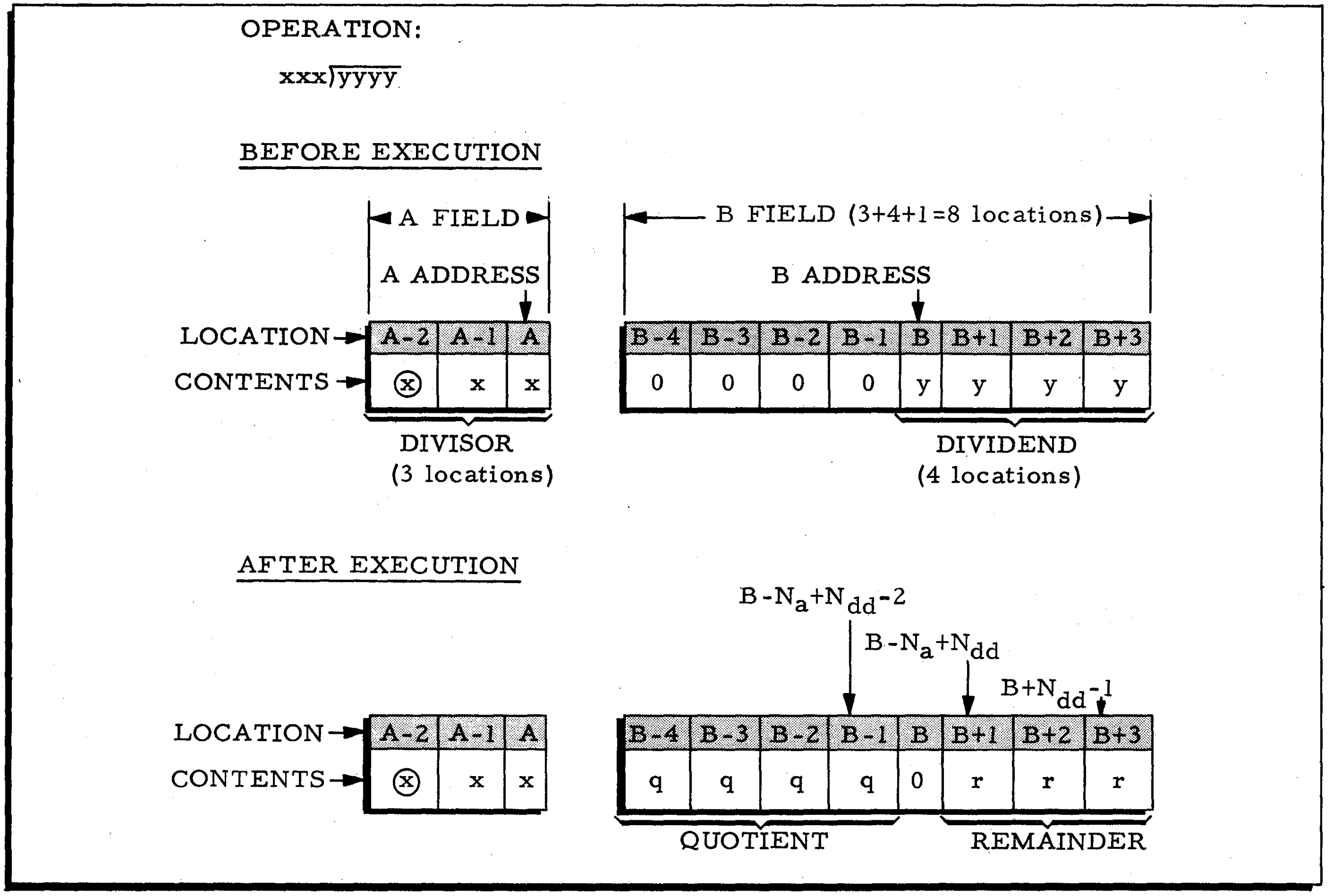


Figure 8-4. Factor Locations in Divide Operation

The leftmost location of the dividend is defined by the B address of the Divide instruction. The rightmost location (i.e., the units position) is the first character location to the right of the B address to have one of its zone bits not equal to zero. As shown in Figure 8-4, all B-field locations to the left of the dividend must contain zeros prior to the divide operation.

A word mark is required in the leftmost location of the divisor. The dividend may or may not contain a word mark.

[1] Note that the B "field" in a divide operation does not define the B operand but is a group of storage locations within which the B operand (the dividend) is contained.

The signs of the operands are indicated by the zone bits in the units positions of the divisor and dividend. Algebraic sign control is used to determine the sign of the quotient (see Table 8-7). The sign of the quotient is expressed in its normalized form (minus = 10, plus = 01). The sign of the remainder is always the same as that of the dividend (in value if not in bit configuration); its form is normalized if the sign of the dividend is normalized.

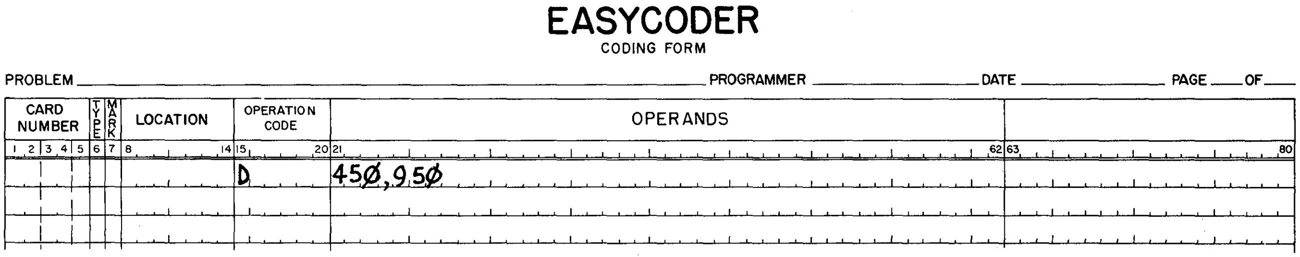
Table 8-7. Divide Sign Conventions

Sign of Divisor	+	+	-	-
Sign of Dividend	+	-	+	-
Sign of Remainder	+	-	+	-
Sign of Quotient	+	-	-	+

Since the presence of a signed digit in the dividend specifies its rightmost location, the units position of the dividend must contain a normalized sign and the zone bits of all other dividend characters must be zero.

When division is completed, the signed decimal quotient is stored in the leftmost locations of the B field; the units position of the quotient is in location B - N_a + N_{dd} - 2, where N_a is the number of locations in the A field and N_{dd} is the number of locations in the dividend. The signed decimal remainder appears in locations B+N_{dd}-1, B+N_{dd}-2, etc. through location B-N_a+N_{dd}. The character location separating the quotient and the remainder is cleared to zero (see Figure 8-4).

In the following example, the divisor is a two-character field whose rightmost location is location 450 and the dividend is a four-character integer whose leftmost location is location 950.

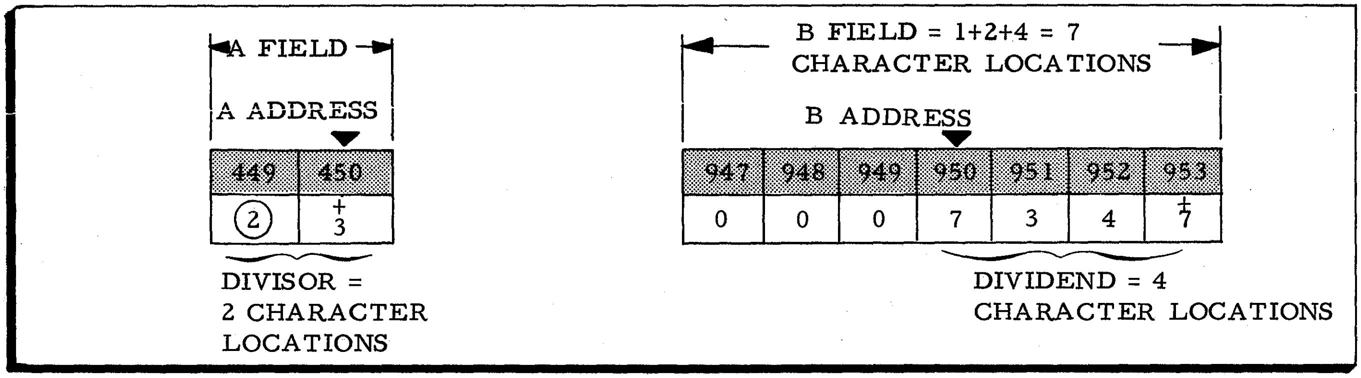


EASYCODER
CODING FORM

PROBLEM ________ PROGRAMMER ________ DATE ________ PAGE ____ OF ____

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					D	45Ø,95Ø	
2							
3							

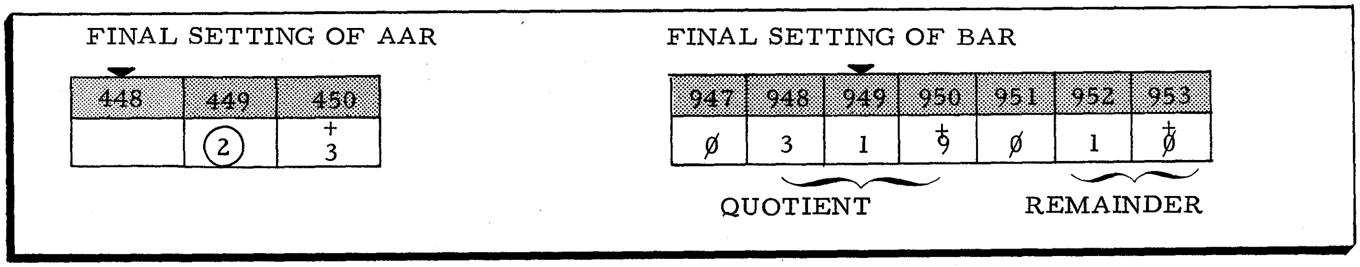
The contents (+23) of the A field are divided into the contents of the field (+7347) whose leftmost location is 950. The rightmost boundary of the dividend is determined by the first character location (location 953) to the right of location B whose zone bits are non-zero. This units position of the dividend therefore contains the sign of the dividend.



449	450
②	$\overset{+}{3}$

947	948	949	950	951	952	953
0	0	0	7	3	4	$\overset{+}{7}$

The quotient (+319) is stored in the leftmost character locations of the B field. The units position of the quotient (location 950) is equal to $B-N_a+N_{dd}-2$, or 950-2+4-2. The remainder is stored in the rightmost locations of the B field; its leftmost location (location 952) is equal to $B-N_a+N_{dd}$, or 950-2+4; its rightmost location (location 953) is equal to $B+N_{dd}-1$, or 950+4-1. The result of the operation is shown below.

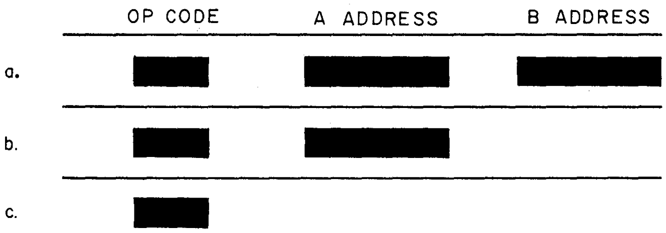


448	449	450
	②	$\overset{+}{3}$

947	948	949	950	951	952	953
Ø	3	1	$\overset{+}{9}$	Ø	1	$\overset{+}{Ø}$

A ADD

FORMAT



FUNCTION

Format a: The signed decimal data in the A field is added algebraically to the signed decimal data in the B field. The result is stored in the B field.

Format b: The signed decimal data in the A field is added to itself. The result is stored in the A field.

Format c: The signed decimal data specified by the contents of the A-address register (AAR) is added algebraically to the signed decimal data specified by the contents of the B-address register (BAR). The result is stored in the B field.

WORD MARKS

Format a: The B operand must have a defining word mark. It is this word mark that terminates the operation. The A operand must have a word mark only if it is shorter than the B operand. In this case, transmission of data from the A operand stops after the A-operand word mark is sensed. If the A field is longer than the B field, the high-order characters of the A field that exceed the field length defined by the B-operand word mark are not processed.

Format b: The A operand must have a defining word mark.

Format c: The B operand must have a defining word mark. The A operand must have a word mark only if it is shorter than the B operand.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	A-N_w	B-N_b
Format b:	NXT	A-N_a	A-N_a
Format c:	NXT	A_p-N_w	B_p-N_b

NOTES

1. The algebraic sign control for the add operation is shown below.

A-FIELD SIGN	+	−	+	−
B-FIELD SIGN	+	−	−	+
TYPE OF ADD	True	True	Comp	Comp
SIGN OF RESULT	Sign of B field		Normalized sign of A or B field, whichever is greater (- = 10, + = 01)	

2. All zone bits in the result field are set to zeros except for the units position (i.e., the sign of the result).

3. This instruction treats both operands as signed decimal data. It will produce ambiguous results if used to manipulate non-decimal data. Particularly, if the four numeric bits of any character have a binary numeric value of 12 or more (octal 14, 15, 16, and 17), the character is treated as if it were a zero, though its zone bits are retained. (In Type 201 or 201-1 processors, the zone and numeric bits of octal 14, 15, 16, and 17 are handled as zeros.) The two remaining cases (octal 12 and 13) are unspecified.

4. The overflow and zero balance indicators are set by an add operation.

5. When the central processor is in the "S" mode of processing, the zone bits are not changed in any character other than the units position of the B field.

EXAMPLE

Add Bond Deductions to Total Deductions.

Description	Tag
Bond Deductions	BDED
Total Deductions	TDED

EASYCODER

CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ____ OF ____

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				A	BDED,TDED	

S SUBTRACT

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	■	■	■
b.	■	■	
c.	■		

FUNCTION

Format a: The signed decimal data in the A field is subtracted algebraically from the signed decimal data in the B field. The result is stored in the B field.

Format b: The signed decimal data in the A field is subtracted from itself. The result is stored in the A field. 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 (with normalized sign).

Format c: The signed decimal data specified by the contents of the A-address register (AAR) is subtracted algebraically from the signed decimal data specified by the contents of the B-address register (BAR). The result is stored in the B field.

WORD MARKS

Format a: The B operand must have a defining word mark. The A operand must have a word mark only if it is shorter than the B operand. In this case, transmission of data from the A operand stops after the A-operand word mark is sensed. If the A field is longer than the B field, the high-order characters of the A field that exceed the field length defined by the B-operand word mark are not processed.

Format b: The A operand must have a defining word mark.

Format c: The B operand must have a defining word mark. The A operand must have a word mark only if it is shorter than the B operand.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	A-N_w	B-N_b
Format b:	NXT	A-N_a	A-N_a
Format c:	NXT	A_p-N_w	B_p-N_b

NOTES

1. Algebraic sign control for the subtract operation is summarized below.

A-FIELD SIGN	+	—	+	—
B-FIELD SIGN	+	—	—	+
TYPE OF ADD	Comp	Comp	True	True
SIGN OF RESULT	Normalized sign of A or B field, whichever is greater (- = 10, + = 01)		Sign of B field	

2. All zone bits in the result field are set to zeros except for the units position (i.e., the sign of the result).

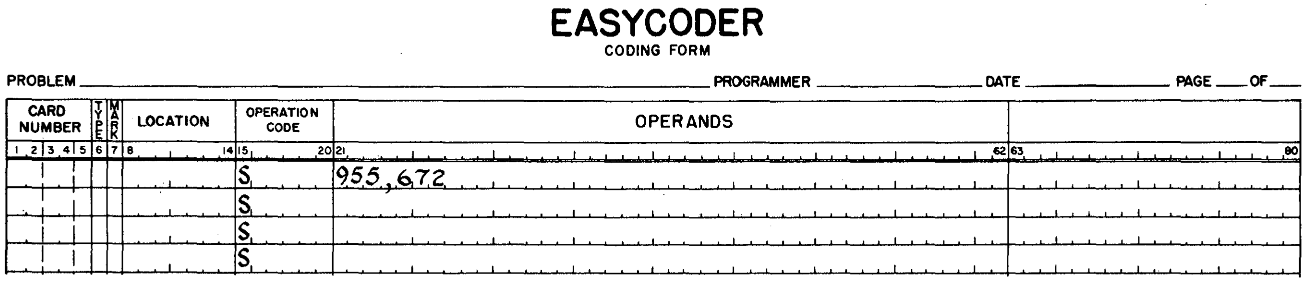
3. This instruction treats both operands as signed decimal data. It will produce ambiguous results if used to manipulate non-decimal data. Particularly, if the four numeric bits of any character have a binary numeric value of 12 or more (octal 14, 15, 16, and 17), the character is treated as if it were a zero, though its zone bits are retained. (In Type 201 or 201-1 processors, the zone and numeric bits of octal 14, 15, 16, and 17 are handled as zeros.) The two remaining cases (octal 12 and 13) are unspecified.

4. The overflow and zero balance indicators are set by a subtract operation.

5. When the central processor is in the "S" mode of processing, the zone bits are not changed in any character other than the units position of the B field.

EXAMPLE

Subtract the contents of the five-character fields starting at location 940, 945, 950, and 955 from the contents of the eight-character fields starting at locations 648, 656, 664, and 672.



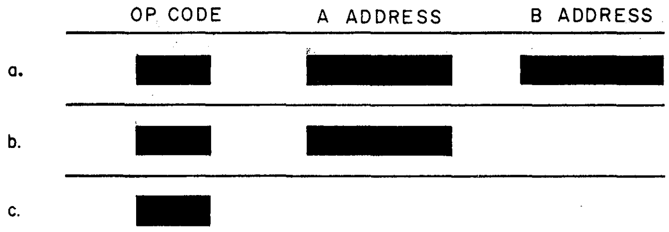
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
1					S	955,672	
2					S		
3					S		
4					S		

BA BINARY ADD

FORMAT



FUNCTION

Format a: The data in the A field is added in binary fashion, character by character, to the data in the B field. The result is stored in the B field.

Format b: The data in the A field is added, character by character, to itself. The result is stored in the A field.

Format c: The data specified by the contents of the A-address register (AAR) is added, character by character, to the data specified by the contents of the B-address register (BAR). The result is stored in the B field.

WORD MARKS

Format a: The B operand must have a defining word mark. It is this word mark that terminates the operation. The A operand must have a word mark only if it is shorter than the B operand. In this case the transmission of data from the A field stops after the A-operand word mark is sensed. If the A field is longer than the B field, the high-order characters of the A field that exceed the field length defined by the B-operand word mark are not processed.

Format b: The A operand must have a defining word mark.

Format c: The B operand must have a defining word mark. The A operand must have a word mark only if it is shorter than the B operand.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	$A-N_w$	$B-N_b$
Format b:	NXT	$A-N_a$	$A-N_a$
Format c:	NXT	A_p-N_w	B_p-N_b

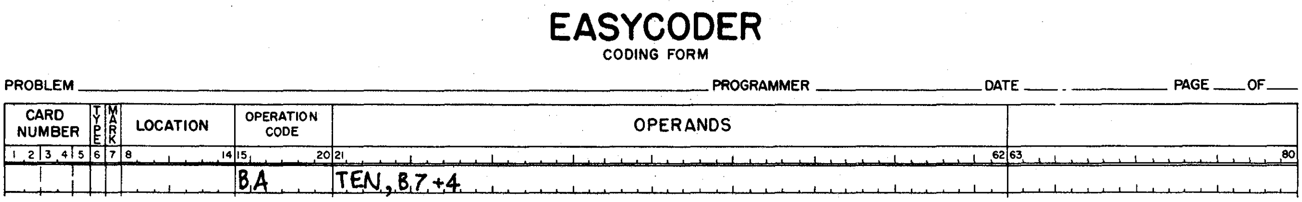
NOTES

1. The overflow and zero balance indicators are not set by a binary add operation.

2. Format b of the BA instruction has the effect of doubling the value stored in the A field; i.e., it shifts the contents of the A field one bit position to the left.

EXAMPLE

Modify the B address of the instruction tagged B7 by the value stored in the location tagged TEN (assuming the use of the two-character addressing mode).



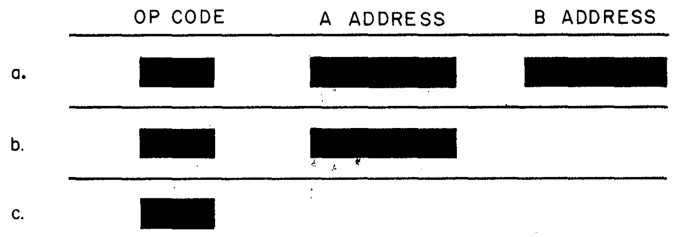
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER (1–5)	TYPE (6)	MARK (7)	LOCATION (8–14)	OPERATION CODE (15–20)	OPERANDS (21–62)	(63–80)
				BA	TEN,B7+4	

BS | BINARY SUBTRACT

FORMAT



FUNCTION

Format a: Each six-bit character in the A field is converted to its ones complement and added, in binary fashion, character by character, to the data in the B field (see page 8-4). A simulated carry is added with the characters in the units position. The result is stored in the B field.

Format b: Each six-bit character in the A field is converted to its ones complement and added, character by character, to itself. A simulated carry is added with the characters in the units position. In effect, this format of the binary subtract instruction replaces the contents of the A field with zeros.

Format c: Each six-bit character specified by the contents of the A-address register (AAR) is converted to its ones complement and added, character by character, to the data specified by the contents of the B-address register (BAR). A simulated carry is added with the characters in the units position. The result is stored in the B field.

WORD MARKS

Format a: The word mark associated with the B operand terminates the operation. The A operand must have a word mark only if it is shorter than the B operand. In this case, transmission of data from the A field stops after the A-operand word mark is sensed. If the A operand is longer than the B operand, the characters of the A operand that exceed the field length defined by the B-operand word mark are not processed.

Format b: The A operand must have a defining word mark.

Format c: The B operand must have a defining word mark. The A operand must have a word mark only if it is shorter than the B operand.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	$A\text{-}N_w$	$B\text{-}N_b$
Format b:	NXT	$A\text{-}N_a$	$A\text{-}N_a$
Format c:	NXT	$A_p\text{-}N_w$	$B_p\text{-}N_b$

NOTES

1. The overflow and zero balance indicators are not set by a binary subtract operation.

2. Formats a. and c. can produce negative results. A negative result is stored in the B field in its twos-complement form. In this case, the absolute numerical value of the result can be obtained by recomplementing the result stored in the B field. A negative result is detected only if the programmer provides appropriate coding to ascertain whether or not operands will produce such a result.

EXAMPLE

Zero the field starting at location TOTAL.

EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				BS	TOTAL	

NOTE: Zone bits as well as numeric bits are cleared to zero by this operation.

ZA	ZERO AND ADD	FEATURES 010 & 011

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	■	■	■
b.	■	■	
c.	■		

FUNCTION

Format a: The data in the A field is transferred, character by character, right to left, to the B field. Zone bits in the B field are set to zero in all positions except the units position. The sign of the result field is based on the sign of the A field (see note 1). If the high-order character of the A field is transferred before the operation terminates, the remaining B field characters are cleared to zeros.

Format b: The data in the A field is converted to an all-numeric format; i.e., the zone bits of all positions in the field except the units position are set to zero. The result remains in the A field. The sign of the A field is not changed by the operation (see note 1).

Format c: The data specified by the contents of the A-address register (AAR) is transferred to the field specified by the contents of the B-address register (BAR). Zone bits in the B field are set to zero in all positions except the units position. The sign of the result field is based on the sign of the A field (see note 1). If the high-order character of the A field is transferred before the operation terminates, the remaining B-field characters are cleared to zeros.

WORD MARKS

Format a: The B operand must have a defining word mark. The A operand must have a word mark only if it is shorter than the B operand. In this case, transfer of data from the A operand stops after the A-operand word mark is sensed. If the A field is longer than the B field, the high-order characters of the A field that exceed the field length defined by the B-operand word mark are not processed.

Format b: The A operand must have a defining word mark.

Format c: The B operand must have a defining word mark. The A operand must have a word mark only if it is shorter than the B operand.

ADDRESS REGISTERS AFTER OPERATION

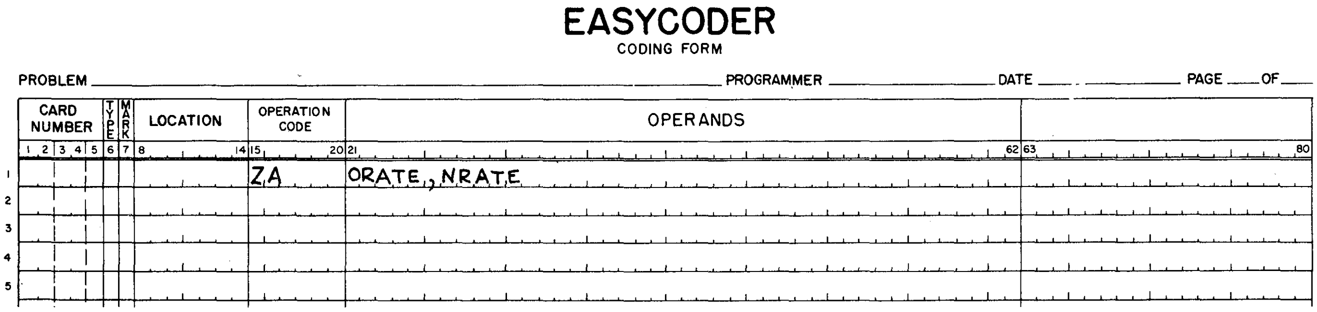
	SR	AAR	BAR
Format a:	NXT	$A-N_w$	$B-N_b$
Format b:	NXT	$A-N_a$	$A-N_a$
Format c:	NXT	A_p-N_w	B_p-N_b

NOTES

1. A plus sign in the units position of the result field is always expressed in its normalized form (01).
2. B-field punctuation is not changed by this operation.
3. This instruction does not set the overflow and zero balance indicators.
4. When the central processor is in the "S" mode of processing and the four numeric bits of any character have a value of 14_8 or more (12_8 in the 4200), the character is treated as if it were a zero. The zero balance indicator is set or reset accordingly.

EXAMPLE

Transfer the contents of the field tagged ORATE to the field tagged NRATE, setting all zone bits in NRATE (except in the units position) to zeros.



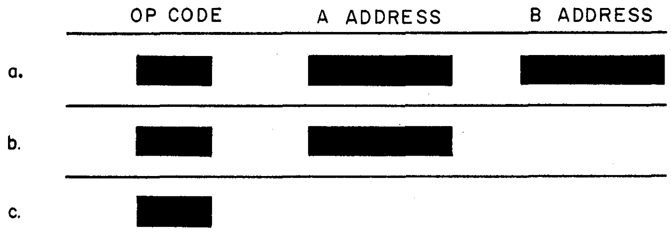
EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ____ OF ____

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					ZA	ORATE,NRATE	
2							
3							
4							
5							

ZS | ZERO AND SUBTRACT | FEATURES 010 & 011

FORMAT



FUNCTION

Format a: The data in the A field is transferred to the B field with the opposite sign. Zone bits in the B field are set to zeros in all positions except the units position. If the high-order character of the A field is transferred before the operation terminates, the remaining B-field characters are cleared to zeros.

Format b: The data in the A field is converted to an all-numeric format; i.e., the zone bits of all positions in the field except the units position are set to zero. The result remains in the A field with its sign reversed.

Format c: The data specified by the contents of the A-address register (AAR) is transferred with the opposite sign to the field specified by the contents of the B-address register (BAR). Zone bits in the B field are set to zero in all positions except the units position. If the high-order character of the A field is transferred before the operation terminates, the remaining B-field characters are cleared to zeros.

WORD MARKS

Format a: The B operand must have a defining word mark. The A operand must have a word mark only if it is shorter than the B operand. In this case, transfer of data from the A operand stops after the A-operand word mark is sensed. If the A field is longer than the B field, the high-order characters of the A field that exceed the field length defined by the B-operand word mark are not processed.

Format b: The A operand must have a defining word mark.

Format c: The B operand must have a defining word mark. The A operand must have a word mark only if it is shorter than the B operand.

ADDRESS REGISTERS AFTER OPERATION

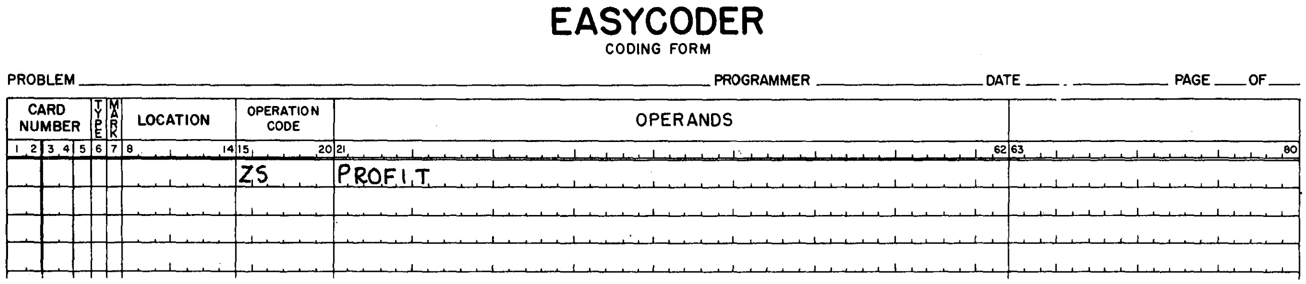
	SR	AAR	BAR
Format a:	NXT	$A\text{-}N_w$	$B\text{-}N_b$
Format b:	NXT	$A\text{-}N_a$	$A\text{-}N_a$
Format c:	NXT	$A_p\text{-}N_w$	$B_p\text{-}N_b$

NOTES

1. A plus sign in the units position of the result field is always expressed in its normalized form (01).
2. B-field punctuation is not changed by this operation.
3. This instruction does not set the overflow and zero balance indicators.
4. When the central processor is in the "S" mode and the four numeric bits of any character have a value of 14_8 or more (12_8 in the 4200), the character is treated as if it were a zero. The zero balance indicator is set or reset accordingly.

EXAMPLE

Change the sign of the data in the field tagged PROFIT.



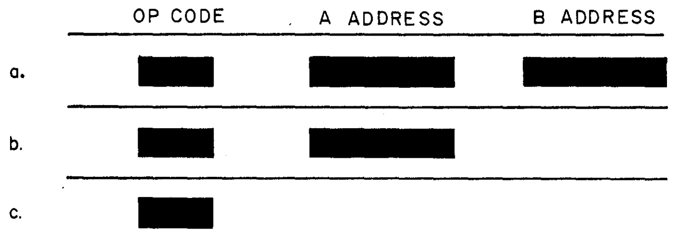
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					ZS	PROFIT	
2							
3							
4							

M MULTIPLY

FORMAT



FUNCTION

Format a: The signed decimal integer in the A field is multiplied by the signed decimal integer in the leftmost locations of the B field. The product is stored, right-justified, in the B field.

Format b: The signed decimal integer in the A field is multiplied by the signed decimal integer in the leftmost locations of the field specified by the contents of the B-address register (BAR). The product is stored, right-justified, in the B field.

Format c: The signed decimal integer in the field specified by the contents of the A-address register (AAR) is multiplied by the signed decimal integer in the leftmost locations of the field specified by the contents of BAR. The product is stored, right-justified, in the B field.

WORD MARKS

Formats a, b, and c:

Word marks are required in the high-order locations of both the A and B fields. All other B-field locations must not contain word marks.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	$A-N_a$	$B-N_b$
Format b:	NXT	$A-N_a$	B_p-N_b
Format c:	NXT	A_p-N_a	B_p-N_b

NOTES

1. The A address of a Decimal Multiply instruction specifies the units position of the multiplicand. The B address specifies a location which is N_a+1 locations to the right of the multiplier, since the B field must contain the multiplier plus enough additional locations (to the right of the multiplier) to provide for the development of the product. Thus, the total number of character locations in the B field must be one greater than the sum of the number of characters in the multiplicand and the multiplier. For example, in a multiplication operation involving a 3-character multiplier and a 5-character multiplicand, 9 positions (5+3+1) must be provided in the B field.

2. Algebraic sign control for the multiply operation is shown below. The sign of the product is expressed in its normalized form (-=10, +=01).

Sign of Multiplicand	+	-	+	-
Sign of Multiplier	+	-	-	+
Sign of Product	+	+	-	-

3. The product is stored (right-justified) in the entire B field, with the unused high-order positions of the B field cleared to zeros. As a result of the operation, the multiplier (initially stored in the B field) is destroyed. Therefore, if the multiplier is to be used more than once, it should be preserved in another storage field.

4. The zero balance indicator is turned ON if the product of the multiply operation is equal to zero; otherwise, the indicator is turned OFF by the operation.

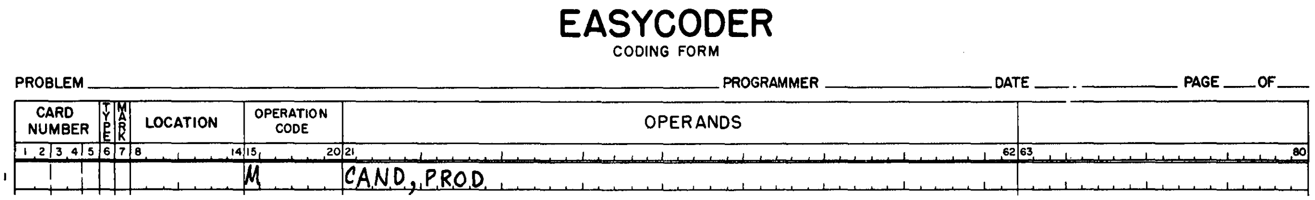
5. This instruction treats both operands as signed decimal data. It will produce ambiguous results if used to manipulate non-decimal data. Particularly, if the four numeric bits of a character have a binary numeric value of 12 or more (octal 14, 15, 16, or 17), the character is treated as if it were a zero. The two remaining cases (octal 12 and 13) are unspecified.

6. This instruction is standard on all processors but the Type 201, on which it is not available.

7. If the A & B operands overlap, then the results are unspecified.

EXAMPLE

Multiply the five-character field tagged CAND by the three-character field whose rightmost character location is six (5+1) less than the location tagged PROD. Store the result, right-justified, in PROD.



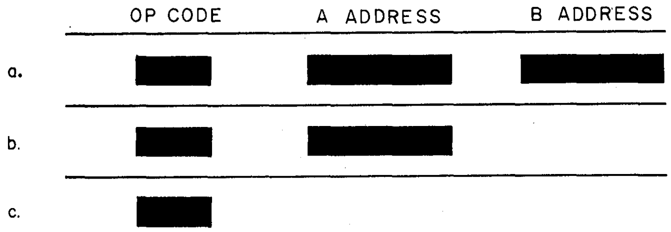
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62	63 ... 80
				M	CAND, PROD	

D DIVIDE

FORMAT



FUNCTION

Format a: The signed decimal integer whose leftmost location is B is divided by the signed decimal integer in the A field. The quotient is stored in the leftmost locations of the B field; the remainder is stored in the rightmost locations of the B field (see page 8-12).

Format b: The signed decimal integer whose leftmost location is specified by the contents of the B-address register (BAR) is divided by the signed decimal integer in the A field. The quotient is stored in the leftmost locations of the B field; the remainder is stored in the rightmost locations of the B field (see page 8-12).

Format c: The signed decimal integer whose leftmost location is specified by the contents of the B-address register (BAR) is divided by the signed decimal integer in the field specified by the contents of the A-address register (AAR). The quotient is stored in the leftmost locations of the B field; the remainder is stored in the rightmost locations of the B field (see page 8-12).

WORD MARKS

Formats a, b, and c:

The A operand (the divisor) must contain a word mark. The B field *may* contain a word mark.

ADDRESS REGISTERS AFTER OPERATION (WHEN DIVISOR IS NOT EQUAL TO ZERO)

	SR	AAR	BAR
Format a:	NXT	$A-N_a$	$B-N_a+N_{dd}-3$
Format b:	NXT	$A-N_a$	$B_p-N_a+N_{dd}-3$
Format c:	NXT	A_p-N_a	$B_p-N_a+N_{dd}-3$

} = Tens position of quotient field

When the divisor is equal to zero, the contents of the address registers are unspecified (see note 1).

NOTES

1. If the divisor is equal to plus or minus zero, the overflow indicator is turned ON, division is not performed, and no memory locations are changed.

2. The length of the B field is determined by adding 1 to the sum of the number of character locations in the divisor and the dividend (B-field length = 1+ length of divisor + length of dividend).

3. The A field (divisor) can be signed or unsigned; if it is unsigned, the divisor is assumed to be positive.

4. The dividend must contain a normalized sign (- = 10, + = 01) in the units position. The zone bits of all other characters in the dividend must be zeros. The proper signing of the dividend is therefore insured if the dividend is moved into the B field by a Zero and Add instruction (see page 8-20).

5. All high-order locations of the B field which are not occupied by the dividend must contain zeros when division begins. These zeros can be automatically inserted if the Zero and Add instruction is used to move the dividend into the B field as mentioned above.

6. The sign of the quotient follows algebraic sign rules as shown below. The sign of the remainder is the original sign of the dividend.

Sign of divisor	+	+	-	-
Sign of dividend	+	-	+	-
Sign of remainder	+	-	+	-
Sign of quotient	+	-	-	+

7. This instruction treats both operands as signed decimal data. It will produce ambiguous results if used to manipulate non-decimal data. Particularly, if the four numeric bits of a character have a binary numeric value of 12 or more (octal 14, 15, 16, or 17), the character is treated as if it were a zero. The two remaining cases (octal 12 and 13) are unspecified.

8. This instruction is standard on all processors but the Type 201, on which it is not available.

9. If the A & B operands overlap, then the results are unspecified.

EXAMPLE

Divide the four-character integer whose leftmost location is location 1000 by the three-character field whose rightmost location is location 500. Store the quotient in the leftmost locations of the field at 1000, and store the remainder in the rightmost locations of this field.

N_a (number of characters in divisor) = 3

N_{dd} (number of characters in dividend) = 4

B (B address) = 1000

Units position of quotient $(B-N_a+N_{dd}-2) = 1000-3+4-2 =$ location 999

Units position of remainder $(B+N_{dd}-1) = 1000+4-1 =$ location 1003

EASYCODER

CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

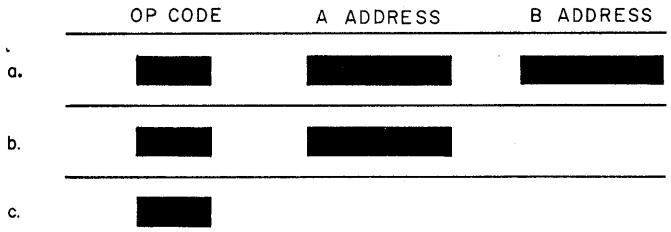
CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				D	5ØØ,1ØØØ	

LOGIC

- EXTRACT
- HALF ADD
- SUBSTITUTE
- COMPARE
- BRANCH
- BRANCH ON CONDITION TEST
- BRANCH ON CHARACTER CONDITION
- BRANCH IF CHARACTER EQUAL
- BRANCH ON BIT EQUAL

EXT | EXTRACT (Logical Product)

FORMAT



FUNCTION

Format a: The data in the A field is combined bit-by-bit with the data in the B field, according to the following rules. The result is stored in the B field.

BIT IN A FIELD	BIT IN B FIELD	BIT IN RESULT FIELD
1	1	1
1	0	0
0	1	0
0	0	0

Format b: The data in the A field is combined bit-by-bit with the data specified by the contents of the B-address register (BAR), according to the rules stated above. The result is stored in the B field.

Format c: The data specified by the contents of the A-address register (AAR) is combined bit-by-bit with the data specified by the contents of BAR, according to the rules stated above. The result is stored in the B field.

WORD MARKS

Formats a, b, and c:

A word mark is required for the shorter of the two operands. The operation terminates when this word mark is sensed.

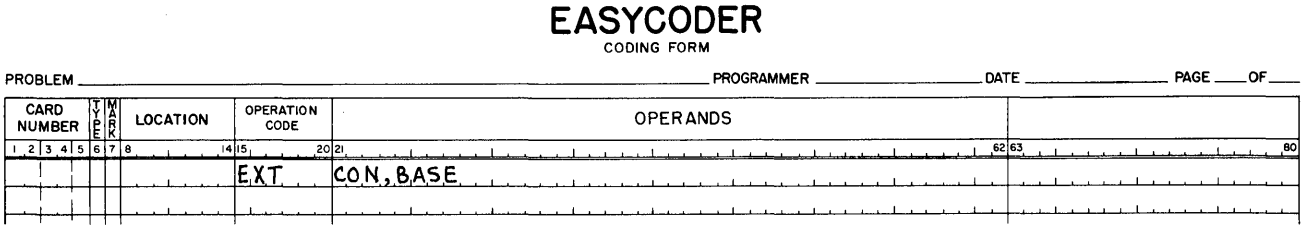
ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	$A-N_w$	$B-N_w$
Format b:	NXT	$A-N_w$	B_p-N_w
Format c:	NXT	A_p-N_w	B_p-N_w

EXAMPLE

Remove all zone bits in the field tagged BASE by combining the contents of BASE with the contents of the field tagged CON. Each character in CON must have the following format:

Bit position	B	A	8	4	2	1
Contents	0	0	1	1	1	1



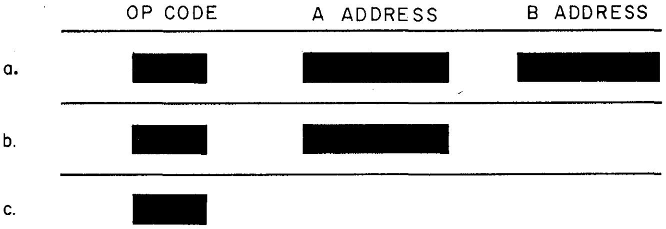
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
1				EXT	CON,BASE	
2						

HA — HALF ADD (Exclusive Or)

FORMAT



FUNCTION

Format a: The data in the A field is combined bit-by-bit with the data in the B field, according to the following rules. The result is stored in the B field.

BIT IN A FIELD	BIT IN B FIELD	BIT IN RESULT FIELD
1	1	0
1	0	1
0	1	1
0	0	0

Format b: The data in the A field is combined bit-by-bit with the data specified by the contents of the B-address register (BAR), according to the rules stated above. The result is stored in the B field.

Format c: The data specified by the contents of the A-address register (AAR) is combined bit-by-bit with the data specified by the contents of BAR, according to the rules stated above. The result is stored in the B field.

WORD MARKS

Formats a, b, and c:

A word mark is required for the shorter of the two operands. The operation terminates when this word mark is sensed.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	$A-N_w$	$B-N_w$
Format b:	NXT	$A-N_w$	B_p-N_w
Format c:	NXT	A_p-N_w	B_p-N_w

EXAMPLE

Clear all the numeric bits in the field tagged SEVEN to zeros by combining the contents of SEVEN with the contents of the field tagged TOO. Do not change the zone bits in SEVEN. (The contents of each character in TOO are 00xxxx, where x equals the corresponding bit in SEVEN.)

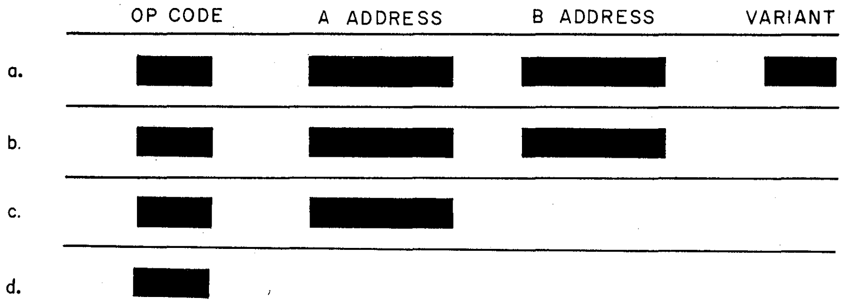
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				HA	TOO, SEVEN	

SST SUBSTITUTE

FORMAT



FUNCTION

Format a: The single character specified by the A address is compared bit-by-bit with the variant character and is moved to the location specified by the B address, according to the following rules:

1. The A-character bit is transferred to the B address if the corresponding variant bit = 1.

2. The B-character bit is preserved if the corresponding variant bit = 0.

Format b: The single character specified by the A address is compared bit-by-bit with the variant character specified in a previous instruction and is moved to the location specified by the B address, according to the rules stated above.

Format c: The single character specified by the A address is compared bit-by-bit with the variant character specified in a previous instruction and is moved to the location specified by the contents of the B-address register (BAR), according to the rules stated above.

Format d: The single character specified by the contents of the A-address register (AAR) is compared bit-by-bit with the variant character specified in a previous instruction and is moved to the location specified by the contents of BAR, according to the rules stated above.

WORD MARKS

Formats a, b, c, and d:

Word marks are not required in either field.

ADDRESS REGISTERS AFTER OPERATION

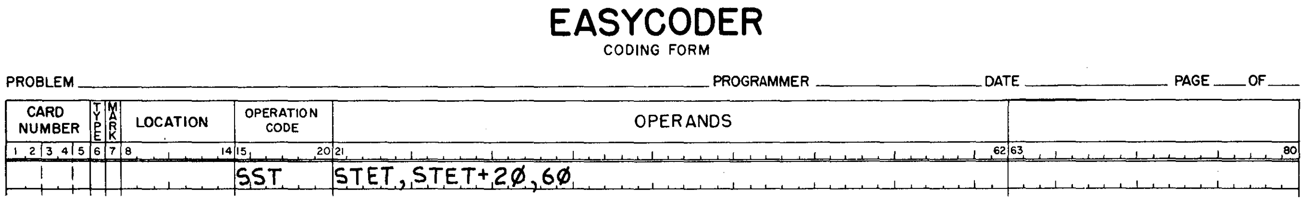
	SR	AAR	BAR
Format a:	NXT	A-1	B-1
Format b:	NXT	A-1	B-1
Format c:	NXT	A-1	B_p-1
Format d:	NXT	A_p-1	B_p-1

NOTE

This instruction can be coded only in formats a. and d. when programming for the Type 201 or 201-1 processor.

EXAMPLES

1. Move the zone bits from the location tagged STET to the location tagged STET +20. A variant character of octal 60 provides the required variant bit configuration (i.e., 110 000).



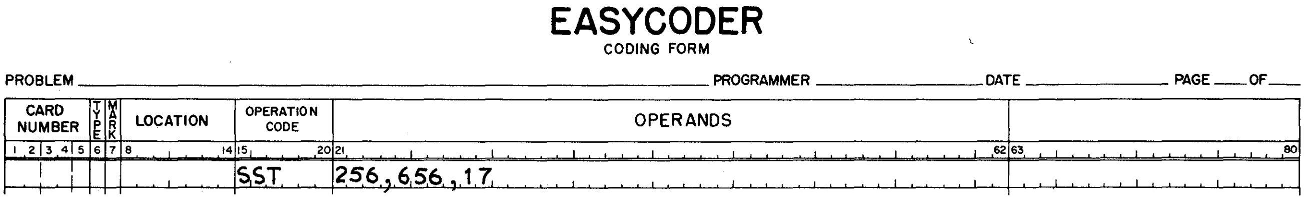
EASYCODER

CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					SST	STET,STET+2Ø,6Ø	

2. Move the numeric portion of the character at location 256 to location 656. A variant of octal 17 provides the required variant bit configuration (i.e., 001 111).



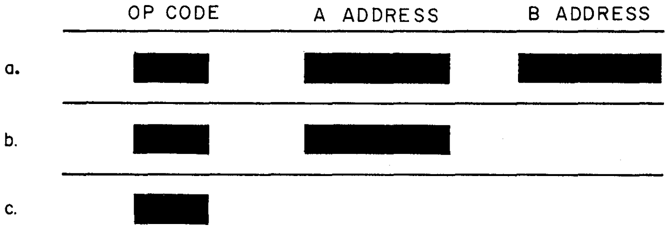
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				SST	256,656,17	

C COMPARE

FORMAT



FUNCTION

Format a: The data in the B field is compared bit-by-bit with the data in the A field. The comparison turns on indicators that can be interrogated by subsequent Branch instructions. The indicators are reset by the next Compare instruction.

Format b: The data specified by the contents of the B-address register (BAR) is compared bit-by-bit with the data in the A field. This operation turns on indicators which can be tested by subsequent Branch instructions. The indicators are reset by the next Compare instruction.

Format c: The data specified by the contents of BAR is compared bit-by-bit with the data in the field specified by the contents of the A-address register (AAR). The comparison turns on indicators that can be interrogated by subsequent Branch instructions. The indicators are reset by the next Compare instruction.

WORD MARKS

Formats a, b, and c:

The word mark associated with the B operand terminates the operation. The A operand must have a word mark only if it is shorter than the B operand. In this case, transmission of data from the A field stops after the A-operand word mark is sensed, and the remaining characters of the B operand are compared with zeros. If the A operand is longer than the B operand, the characters of the A operand that exceed the field length defined by the B-operand word mark are not processed.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	$A-N_w$	$B-N_b$
Format b:	NXT	$A-N_w$	B_p-N_b
Format c:	NXT	A_p-N_w	B_p-N_b

NOTES

1. All characters that can appear in storage can be compared. The ascending order of characters is listed in Appendix B.
2. Both fields must have exactly the same bit configurations to be equal. For example, plus zero is not equal to minus zero.
3. Comparison results and associated branch conditions are listed below.

COMPARISON RESULT	BRANCH CONDITION
$B<A$	Low Compare
$B=A$	Equal Compare
$B\leq A$	Low or Equal Compare
$B>A$	High Compare
$B\neq A$	Unequal Compare
$B\geq A$	High 'or Equal Compare

EXAMPLE

Compare item number with 4000. If item number equals 4000, continue the program in sequence; otherwise, branch to location NITEM.

Description	Tag
Item Number	ITEM
4000	CON4

EASYCODER

CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					C	CON4, ITEM	
2					BCT	NITEM, 45	

B	BRANCH

FORMAT

OP CODE	A ADDRESS	B ADDRESS	VARIANT
■	■		

FUNCTION

The Branch instruction causes the program to branch to the location specified by the A address and to store the contents of the sequence register (SR) in the B-address register (BAR). It is used to interrupt normal program sequence and to continue the program at any desired point, without testing for specific conditions. Thus, this instruction is frequently referred to as an "unconditional branch."

WORD MARKS

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR
JI (A)	A	NXT

NOTES

1. The A address is placed in AAR during the extraction of this instruction, preserving any active high-order bits in AAR. When the instruction is executed, the entire contents of AAR specify the address to which the program branches. Also, the entire contents of SR are stored in BAR during the execution phase.

2. The contents of the variant register are unspecified following the execution of this instruction. Therefore an instruction requiring a variant character must not be chained following a Branch instruction.

EXAMPLE

Select the next instruction from the location tagged SUB6.

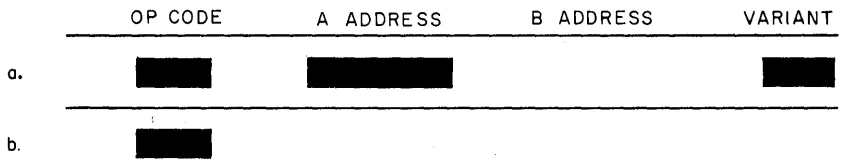
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
1					B	SUB6	
2							
3							
4							
5							
6							
7							
8							

BCT | BRANCH ON CONDITION TEST

FORMAT



FUNCTION

Format a: The variant character specifies a condition indicator or a SENSE switch to be tested. If the condition being tested is present, the program branches to the location specified by the A address and the contents of the sequence register (SR) are stored in the B-address register (BAR). If the condition specified by the variant character is not present, the program continues in sequence. Tables 8-8 and 8-9 list the valid variant characters and the conditions they test.

Format b: If the condition specified by the previous variant character is present, the program branches to the location specified by the contents of the A-address register (AAR) and the contents of SR are stored in BAR. If the condition being tested is not present, the program continues in sequence. Tables 8-8 and 8-9 list the valid variant characters and the conditions they test.

WORD MARKS

Formats a and b:

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR	
Format a:	JI (A)	A	NXT	BRANCH
	NXT	A	B_p	NO BRANCH
Format b:	JI (A_p)	A_p	NXT	BRANCH
	NXT	A_p	B_p	NO BRANCH

NOTES

1. If the overflow indicator is tested and an overflow condition exists, the indicator is automatically reset as a result of being tested. In all other cases, the indicator tested is not reset as a result of the test.

2. The comparison indicators are:

   a. set by the Compare instruction;

   b. set by the Table Lookup instruction;

   c. stored (and cleared) by the Store Variant and Indicators instruction;

Table 8-8. SENSE Switch Test Conditions for BCT Instruction

Variant Character (Octal)	Branch On
00	Unconditional
01	SENSE Switch 1 On
02	SENSE Switch 2 On
03	SENSE Switches 1 and 2 On
04	SENSE Switch 3 On
05	SENSE Switches 1 and 3 On
06	SENSE Switches 2 and 3 On
07	SENSE Switches 1, 2, and 3 On
10	SENSE Switch 4 On
11	SENSE Switches 1 and 4 On
12	SENSE Switches 2 and 4 On
13	SENSE Switches 1, 2, and 4 On
14	SENSE Switches 3 and 4 On
15	SENSE Switches 1, 3, and 4 On
16	SENSE Switches 2, 3, and 4 On
17	SENSE Switches 1, 2, 3, and 4 On
20	Unconditional
21	SENSE Switch 5 On
22	SENSE Switch 6 On
23	SENSE Switches 5 and 6 On
24	SENSE Switch 7 On
25	SENSE Switches 5 and 7 On
26	SENSE Switches 6 and 7 On
27	SENSE Switches 5, 6, and 7 On
30	SENSE Switch 8 On
31	SENSE Switches 5 and 8 On
32	SENSE Switches 6 and 8 On
33	SENSE Switches 5, 6, and 8 On
34	SENSE Switches 7 and 8 On
35	SENSE Switches 5, 7, and 8 On
36	SENSE Switches 6, 7, and 8 On
37	SENSE Switches 5, 6, 7, and 8 On

NOTE: When testing for a multiple SENSE switch condition, a branch occurs only if all of the specified conditions are met.

Table 8-9. Indicator Test Conditions for BCT Instruction

Variant Character (Octal)	Branch On
40	Do not branch
41	B<A (Low Compare)
42	B=A (Equal Compare)
43	B≤A (Low or Equal Compare)
44	B>A (High Compare)
45	B≠A (Unequal Compare)
46	B≥A (High or Equal Compare)
47	Unconditional
50	Overflow
51	Overflow or B<A
52	Overflow or B=A
53	Overflow or B≤A
54	Overflow or B>A
55	Overflow or B≠A
56	Overflow or B≥A
57	Unconditional
60	Zero Balance
61	Zero Balance or B<A
62	Zero Balance or B=A
63	Zero Balance or B≤A
64	Zero Balance or B>A
65	Zero Balance or B≠A
66	Zero Balance or B≥A
67	Unconditional
70	Overflow or Zero Balance
71	Overflow or Zero Balance or B<A
72	Overflow or Zero Balance or B=A
73	Overflow or Zero Balance or B≤A
74	Overflow or Zero Balance or B>A
75	Overflow or Zero Balance or B≠A
76	Overflow or Zero Balance or B≥A
77	Unconditional

NOTE: When testing for a multiple indicator condition, a branch occurs if any one of the specified conditions is met.

d. restored by the Restore Variant and Indicators instruction;

e. restored by the Resume Normal Mode instruction if coming out of the external interrupt mode (but not out of internal interrupt mode);

f. stored when an external interrupt occurs.

3. The A address (if any) is placed in AAR during the extraction of this instruction, preserving any active high-order bits in AAR. If the instruction causes a branch (i.e., if the condition being tested is present), the entire contents of AAR specify the address to which the program branches when the instruction is executed. Also, the entire contents of SR are stored in BAR during the execution phase of the instruction.

4. Consider the variant character in its six-bit form $V_6V_5V_4V_3V_2V_1$. The following chart may be used to determine the variant character to be used in a BCT instruction.

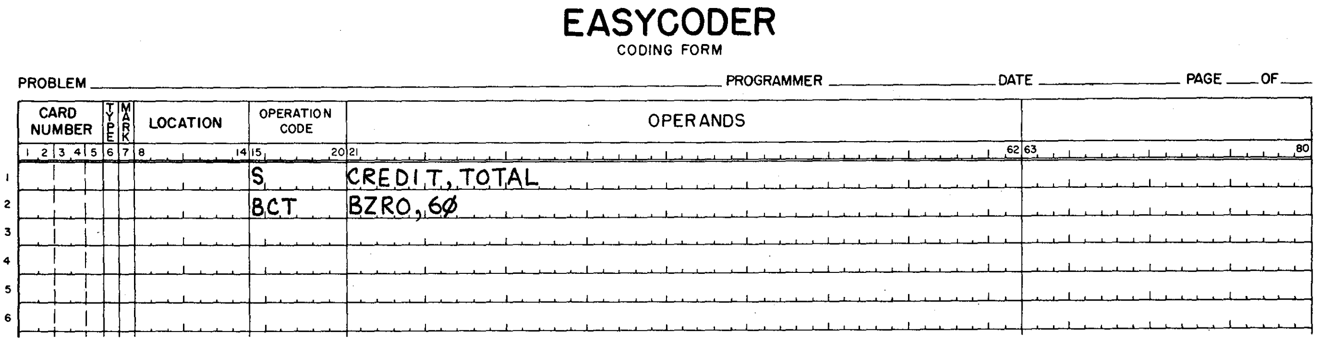
V_6	V_5	V_4	V_3	V_2	V_1
00 = Test SENSE Switches 1 through 4		SENSE Switch 4	SENSE Switch 3	SENSE Switch 2	SENSE Switch 1
01 = Test SENSE Switches 5 through 8		SENSE Switch 8	SENSE Switch 7	SENSE Switch 6	SENSE Switch 5
1 = Test Zero Balance, Overflow, or Compare	Zero Balance	Overflow	High Compare	Equal Compare	Low Compare

5. SENSE switches 5 through 8 are included as a standard feature with the Type 2201 and 4201 processors and are not available with the Model 200, 1200, or 1250 processors.

6. This instruction can be coded only in format a. when programming for the Type 201 or 201-1 processor.

EXAMPLE

Subtract CREDIT from TOTAL and test for a zero balance. If this condition exists branch to BZRO; otherwise continue the program in sequence.



EASYCODER

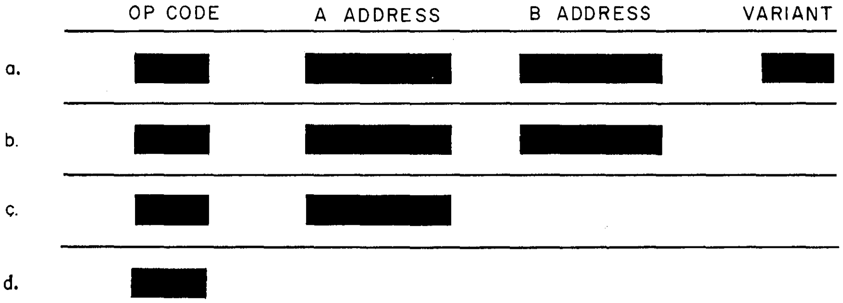
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ___ OF ___

	CARD NUMBER (1–5)	TYPE (6)	MARK (7)	LOCATION (8–14)	OPERATION CODE (15–20)	OPERANDS (21–62)	(63–80)
1					S	CREDIT,TOTAL	
2					BCT	BZRO,6Ø	
3							
4							
5							
6							

BCC — BRANCH ON CHARACTER CONDITION

FORMAT



FUNCTION

Format a: The single character specified by the B address is examined for the condition specified by the variant character. If the condition is present, the program branches to the location specified by the A address, and the contents of the sequence register (SR) are stored in the B-address register (BAR). If the condition is not present, the program continues in sequence. The valid variant characters and the condition each represents are listed in Tables 8-10 and 8-11.

Format b: The single character specified by the B address is examined for the condition specified by the variant character of a previous instruction. If the condition is present, the program branches to the location specified by the A address, and the contents of SR are stored in BAR. If the condition is not present, the program continues in sequence. The valid variant characters and the condition each represents are listed in Tables 8-10 and 8-11.

Format c: The single character specified by the contents of BAR is examined for a condition specified by the variant character of a previous instruction. If the condition is present, the program branches to the location specified by the A address, and the contents of SR are stored in BAR. If the condition is not present, the program continues in sequence. The valid variant characters and the condition each represents are listed in Tables 8-10 and 8-11.

Format d: The single character specified by the contents of BAR is examined for a condition specified by the variant character of a previous instruction. If the condition is present, the program branches to the location specified by the contents of the A-address register (AAR), and the contents of SR are stored in BAR. If the condition is not present, the program continues in sequence. The valid variant characters and the condition each represents are listed in Tables 8-10 and 8-11. Series 200 processors which are equipped with the advanced programming instructions (see Table 1-11, page 1-18) can interpret any bit configuration of the variant character, ranging from octal 00 to octal 77. The valid variant characters which can be interpreted with this option are shown in Table 8-11 and expanded in Appendix B.

Table 8-10. Basic Test Conditions for BCC Instruction

Variant Character (Octal)	Character Condition
00	Unconditional
02	The B bit of the character at B is 1.
06	The character at B contains a negative sign (the B and A bits are 10).
10	The character at B contains either a word mark or a record mark (the word-mark bit is 1).
12	The B bit is 1 <u>and</u> the word-mark bit is 1.
16	The character at B contains a negative sign <u>and</u> the word-mark bit is 1.
20	The character at B contains either an item mark or a record mark (the item-mark bit is 1).
22	The B bit is 1 <u>and</u> the item-mark bit is 1.
26	The character at B contains a negative sign <u>and</u> the item-mark bit is 1.
30	The character at B contains a record mark (the word-mark and item-mark bits are 11).
32	The character at B contains a record mark <u>and</u> the B bit is 1.
36	The character at B contains a record mark <u>and</u> a negative sign.

WORD MARKS

Formats a, b, c, and d:

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR	
Format a:	JI (A)	A	NXT	BRANCH
	NXT	A	B-1	NO BRANCH
Format b:	JI (A)	A	NXT	BRANCH
	NXT	A	B-1	NO BRANCH
Format c:	JI (A)	A	NXT	BRANCH
	NXT	A	B_p-1	NO BRANCH
Format d:	JI (A_p)	A_p	NXT	BRANCH
	NXT	A_p	B_p-1	NO BRANCH

Table 8-11. BCC Test Conditions with Advanced Programming Instructions

Variant Character (Octal)	Character Condition
X0	No condition.
X1	The A bit of the character at B is 1.
X2	The B bit of the character at B is 1.
X3	The B and A bits of the character at B are 11.
X4	The B and A bits of the character at B are 00.
X5	The character at B contains a positive sign (the B and A bits are 01).
X6	The character at B contains a negative sign (the B and A bits are 10).
X7	The B and A bits of the character at B are 11 (same as X3 above).
0X	No condition.
1X	The word-mark bit of the character at B is 1 (either a word mark or a record mark is present).
2X	The item-mark bit of the character at B is 1 (either an item mark or a record mark is present).
3X	The character at B contains a record mark.
4X	The character at B contains no punctuation mark.
5X	The character at B contains a word mark only, not an item mark.
6X	The character at B contains an item mark only, not a word mark.
7X	This is a special case; see note 2.

NOTES:

1. An X represents any octal digit. If both octal digits specify "no condition" (i.e., 00), the branch occurs unconditionally. If only one digit is 0, the branch occurs if the condition specified by the other digit is met. If both octal digits specify conditions, the branch occurs if both conditions are met. The variant character 7X is an exception to these rules, as described in note 2.

2. The Type 201 and 201-1 processors interpret a 7X variant as if it were a 3X (i.e., branch to the A address if the character at B contains a record mark and the condition specified by X is met).

   All other processors interpret the 7X variant as follows:

   a. If X is 0, the branch is an unconditional branch.

   b. If X is any digit other than 0, the branch occurs if either the condition specified by the rightmost digit is met or the character at B contains a word mark.

NOTES

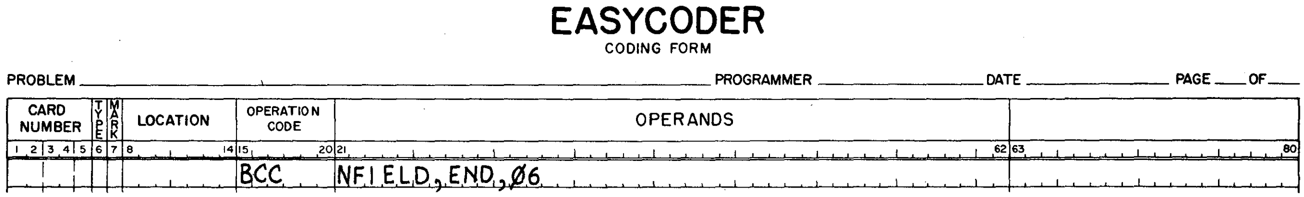
1. If the octal configuration of the variant character is 00 or 70, the branch is unconditional.

2. The A address (if any) is placed in AAR during the extraction of the BCC instruction, preserving any active high-order bits in AAR. If the instruction causes a branch (i.e., if the condition being tested is present), the entire contents of AAR specify the address to which the program branches when the instruction is executed. Also, the entire contents of SR are placed in BAR during the execution phase.

3. This instruction can be coded only in formats a. and d. when programming for the Type 201 or 201-1 processor.

EXAMPLE

If the location tagged END contains a negative sign, branch to the location tagged NFIELD. Otherwise, continue the program in sequence.



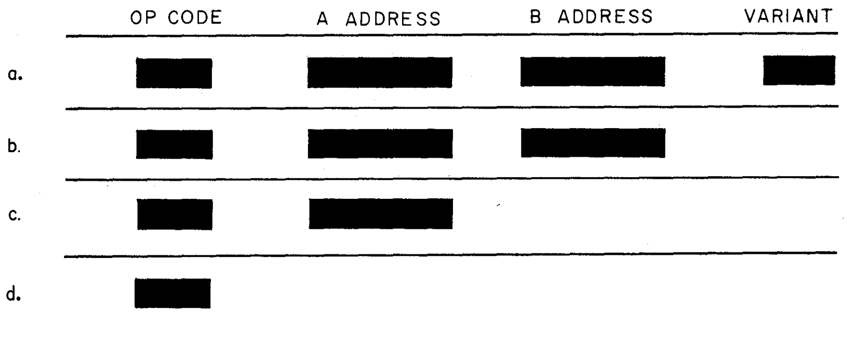
EASYCODER
CODING FORM

PROBLEM ________ PROGRAMMER ________ DATE ________ PAGE ___ OF ___

CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
				BCC	NFIELD,END,Ø6	

BCE | BRANCH IF CHARACTER EQUAL | FEATURES 010 & 011

FORMAT



FUNCTION

Format a: The single character specified by the B address is compared to the variant character. If the bit configurations of the two characters are equal, the program branches to the location specified by the A address, and the contents of the sequence register (SR) are stored in tbe B-address register (BAR). If the bit configurations are unequal, the program continues in sequence.

Format b: The single character specified by the B address is compared to the variant character specified in a previous instruction. If the bit configurations of the two characters are equal, the program branches to the location specified by the A address, and the contents of SR are stored in BAR. If the bit configurations are unequal, the program continues in sequence.

Format c: The single character specified by the contents of BAR is compared to the variant character specified in a previous instruction. If the bit configurations of the two characters are equal, the program branches to the location specified by the A address, and the contents of SR are stored in BAR. If the bit configurations are unequal, the program continues in sequence.

Format d: The single character specified by the contents of BAR is compared to the variant character specified in a previous instruction. If the bit configurations of the two characters are equal, the program branches to the location specified by the contents of the A-address register (AAR), and the contents of SR are stored in BAR. If the bit configurations are unequal, the program continues in sequence.

WORD MARKS

Formats a, b, c, and d:

A word mark in the location tested has no effect on the instruction.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR	
Format a:	JI (A)	A	NXT	BRANCH
	NXT	A	B-1	NO BRANCH
Format b:	JI (A)	A	NXT	BRANCH
	NXT	A	B-1	NO BRANCH
Format c:	JI (A)	A	NXT	BRANCH
	NXT	A	B_p-1	NO BRANCH
Format d:	JI (A_p)	A_p	NXT	BRANCH
	NXT	A_p	B_p-1	NO BRANCH

NOTES

1. This instruction can be coded only in formats a. and d. when programming for the Type 201 or 201-1 processor.

2. The A address (if any) is placed in AAR during the extraction of the BCE instruction, preserving any active high-order bits in AAR. If the instruction causes a branch (i.e., if the condition being tested is present), the entire contents of AAR specify the address to which the program branches when the instruction is executed. Also, the entire contents of SR are placed in BAR during the execution phase.

3. When the central processor is in the "S" mode, execution of the BCE instruction sets the comparison indicators to show whether B>V, B=V, or B<V.

EXAMPLES

1. Determine if the character stored in the location tagged LABEL+3 is equal to 6. If so, branch to the location tagged P6; otherwise continue the program in sequence.

EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER 1 2 3 4 5	TYPE 6	MARK 7	LOCATION 8 - 14	OPERATION CODE 15 - 20	OPERANDS 21 - 62	63 - 80
1					BCE	P6,LABEL+3,6	

2. Determine if any character position in the seven-character field tagged PART contains the letter Q. If so, branch to the location tagged RETRO; otherwise continue the program in sequence.

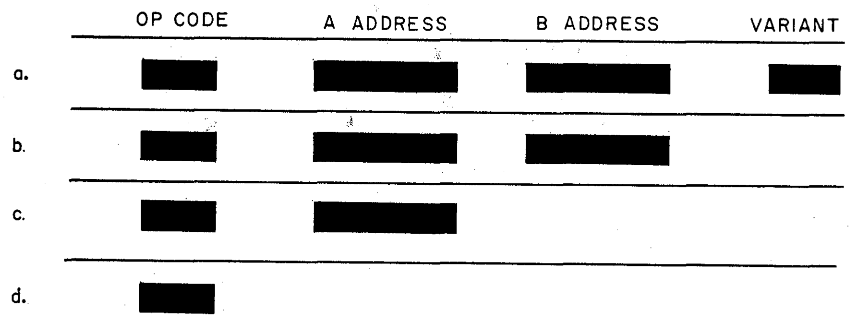
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER 1 2 3 4 5	TYPE 6	MARK 7	LOCATION 8 - 14	OPERATION CODE 15 - 20	OPERANDS 21 - 62	63 - 80
1					BCE	RETRO,PART,Q	
2					BCE		
3					BCE		
4					BCE		
5					BCE		
6					BCE		
7					BCE		

BBE BRANCH ON BIT EQUAL FEATURE 010

FORMAT



FUNCTION

Format a: The single character specified by the B address is combined bit-by-bit with the variant character, according to the rules shown below. If the result (the logical product) is not equal to zero, the program branches to the location specified by the A address, and the contents of the sequence register (SR) are stored in the B-address register (BAR). If the result is equal to zero, the program continues in sequence.

Bit in B Character	Bit in Variant Character	Bit in Result Field
1	1	1
1	0	0
0	1	0
0	0	0

Format b: The single character specified by the B address is combined bit-by-bit with the variant character specified in a previous instruction, according to the rules shown above. If the result is not equal to zero, the program branches to the location specified by the A address, and the contents of SR are stored in BAR. If the result is equal to zero, the program continues in sequence.

Format c: The single character specified by the contents of BAR is combined bit-by-bit with the variant character specified in a previous instruction, according to the rules shown above. If the result is not equal to zero, the program branches to the location specified by the A address, and the contents of SR are stored in BAR. If the result is equal to zero, the program continues in sequence.

Format d: The single character specified by the contents of BAR is combined bit-by-bit with the variant character specified in a previous instruction, according to the rules shown above. If the result is not equal to zero, the program branches to the location specified by the contents of the A-address register (AAR), and the contents of SR are stored in BAR. If the result is equal to zero, the program continues in sequence.

WORD MARKS

Formats a, b, c, and d:

Word marks are not tested by this instruction and have no effect on the operation.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR	
Format a:	JI (A)	A	NXT	BRANCH
	NXT	A	B-1	NO BRANCH
Format b:	JI (A)	A	NXT	BRANCH
	NXT	A	B-1	NO BRANCH
Format c:	JI (A)	A	NXT	BRANCH
	NXT	A	B_p-1	NO BRANCH
Format d:	JI (A_p)	A_p	NXT	BRANCH
	NXT	A_p	B_p-1	NO BRANCH

NOTES

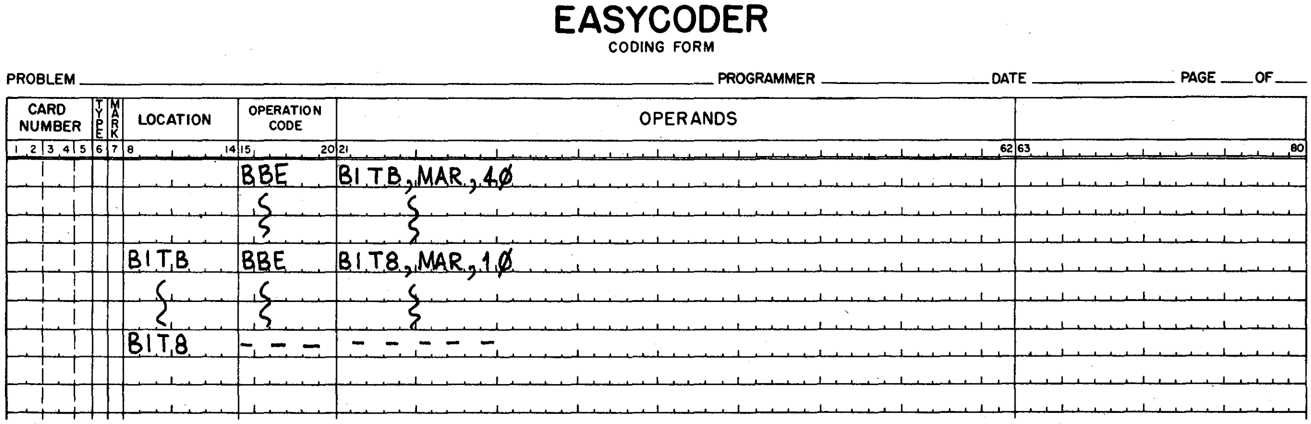
1. The logical product formed by this instruction is tested but is not stored. Main memory locations are not disturbed by this operation.

2. The A address (if present) is placed in AAR during the extraction of the instruction, preserving any active high-order bits in AAR. If the instruction causes a branch (i.e., if the logical product does not equal zero), the entire contents of AAR specify the address to which the program branches when the instruction is executed. Also, the entire contents of SR are placed in BAR during the execution phase.

3. Since this instruction results in a branch if any bit product is not equal to zero, only one bit at a time should be tested. Other bits can be checked by branching to additional BBE instructions.

EXAMPLE

Branch to the location tagged BIT8 only if the character at the location tagged MAR contains a 1 in both the B and the 8 bit positions. Otherwise, continue in sequence. This example requires two BBE instructions to test the two bits in question; location BIT8 is reached only if both tests are met.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

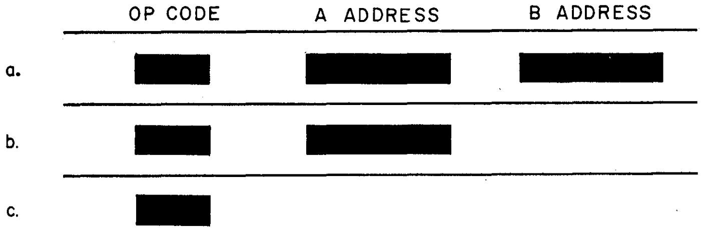
	CARD NUMBER 1 2 3 4 5	TYPE 6	MARK 7	LOCATION 8–14	OPERATION CODE 15–20	OPERANDS 21–62	63–80
1					BBE	BITB,MAR,4Ø	
2					≀	≀	
3					≀	≀	
4				BITB	BBE	BIT8,MAR,1Ø	
5				≀	≀	≀	
6				≀	≀	≀	
7				BIT8	- - -	- - - - -	
8							
9							

CONTROL

- SET WORD MARK
- SET ITEM MARK
- CLEAR WORD MARK
- CLEAR ITEM MARK
- HALT
- NO OPERATION
- MOVE CHARACTERS TO WORD MARK
- LOAD CHARACTERS TO A-FIELD WORD MARK
- STORE CONTROL REGISTERS
- LOAD CONTROL REGISTERS
- CHANGE ADDRESSING MODE
- CHANGE SEQUENCING MODE
- EXTENDED MOVE
- MOVE AND TRANSLATE
- MOVE ITEM AND TRANSLATE
- LOAD INDEX/BARRICADE INDICATOR
- STORE INDEX/BARRICADE INDICATOR
- TABLE LOOKUP
- MOVE OR SCAN

SW | SET WORD MARK

FORMAT



FUNCTION

Format a: A word mark is set at the location specified by each address. The data and item-mark bits at each location are undisturbed.

Format b: A word mark is set at the location specified by the A address. The data and item-mark bits at this location are undisturbed.

Format c: Word marks are set at the locations specified by the contents of the A- and B-address registers (AAR and BAR). The data and item-mark bits at each location are undisturbed.

WORD MARKS

Formats a, b, and c:

Word marks are set as described above.

ADDRESS REGISTERS AFTER OPERATION

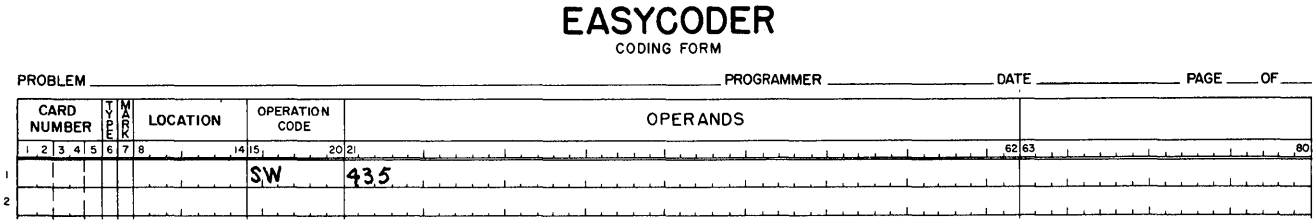
	SR	AAR	BAR
Format a:	NXT	A-1	B-1
Format b:	NXT	A-1	A-1
Format c:	NXT	A_p-1	B_p-1

NOTE

The extraction of this instruction when coded in format a. automatically terminates when the last character of the B address is loaded into BAR. Therefore, a word mark is not required in the location following the B address.

EXAMPLE

Set a word mark in location 435.



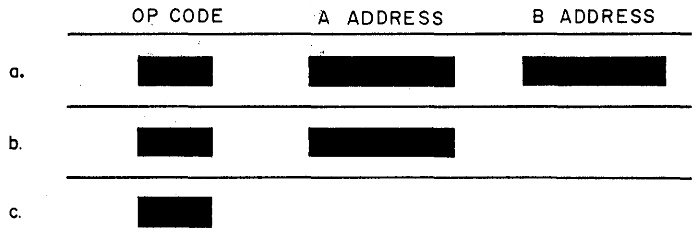
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				SW	435	
2						

SI SET ITEM MARK

FORMAT



FUNCTION

Format a: An item mark is set at the location specified by each address. The data and word-mark bits at each location are undisturbed.

Format b: An item mark is set at the location specified by the A address. The data and word-mark bits at this location are undisturbed.

Format c: Item marks are set at the locations specified by the contents of the A- and B-address registers (AAR and BAR). The data and word-mark bits at each location are undisturbed.

WORD MARKS

Formats a, b, and c:

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	A-1	B-1
Format b:	NXT	A-1	A-1
Format c:	NXT	A_p-1	B_p-1

NOTE

The extraction of this instruction when coded in format a. automatically terminates when the last character of the B address is loaded into BAR. Therefore, a word mark is not required in the location following the B address.

EXAMPLE

Set item marks in the locations tagged ENT and ENT+80

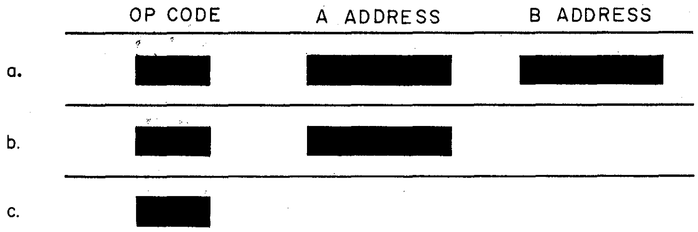
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER 1 2 3 4 5	TYPE 6	MARK 7	LOCATION 8-14	OPERATION CODE 15-20	OPERANDS 21-62	63-80
				SI	ENT, ENT+8Ø	

CW CLEAR WORD MARK

FORMAT



FUNCTION

Format a: The locations specified by the A and B addresses are cleared of word marks. The data and item-mark bits at these locations are undisturbed.

Format b: The word mark at the location specified by the A address is cleared. The data and item-mark bits at this location are undisturbed.

Format c: Word marks are cleared at the locations specified by the contents of the A- and B-address registers (AAR and BAR). The data and item-mark bits at these locations are undisturbed.

WORD MARKS

Formats a, b, and c:

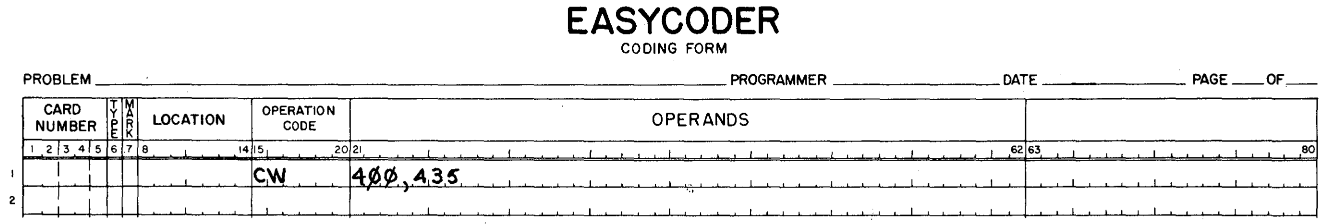
Word marks are cleared as defined above.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	A-1	B-1
Format b:	NXT	A-1	A-1
Format c:	NXT	A_p-1	B_p-1

EXAMPLE

Clear the word marks at locations 400 and 435.



EASYCODER

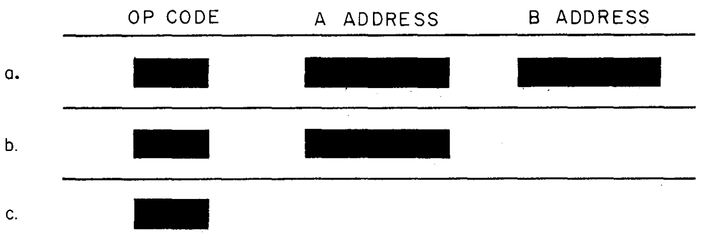
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				CW	4ØØ,435	

CI CLEAR ITEM MARK

FORMAT



FUNCTION

Format a: Item marks are cleared from the locations specified in the A and B addresses. The data and word-mark bits at these locations are undisturbed.

Format b: The item mark at the location specified by the A address is cleared. The data and word-mark bits at this location are undisturbed.

Format c: Item marks are cleared at the locations specified by the contents of the A- and B-address registers (AAR and BAR). The data and word-mark bits at these locations are undisturbed.

WORD MARKS

Formats a, b, and c:

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	A-1	B-1
Format b:	NXT	A-1	A-1
Format c:	NXT	A_p-1	B_p-1

EXAMPLE

Clear the item mark in location REC.

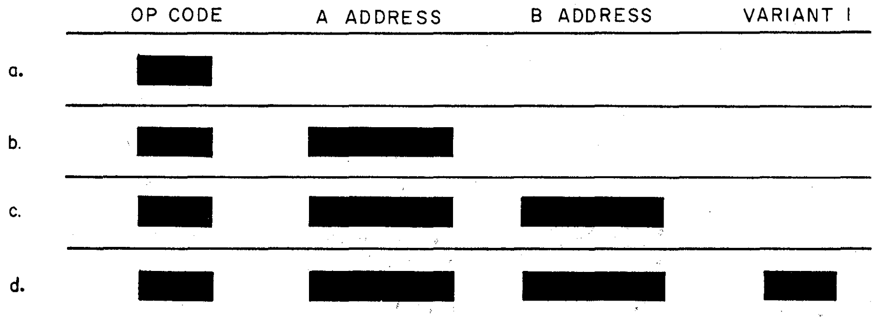
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				CI	REC	

H HALT

FORMAT



FUNCTION

Format a: This instruction causes the machine to stop. Pressing the RUN button causes the program to resume with the next instruction in sequence.

Format b: The contents of the sequence register (SR) are stored in the B-address register (BAR); the A address of the instruction is transferred to SR; then the machine stops. Pressing the RUN button causes the program to resume with the instruction specified in the A address. This format is usually referred to as a "halt and branch" instruction.

Format c: This instruction causes the machine to stop. Pressing the RUN button causes the program to resume with the next instruction in sequence. The address portions can be used to indicate control information such as a halt identification number (see note 2).

Format d: This instruction causes the machine to stop. Pressing the RUN button causes the program to resume with the next instruction in sequence. The address portions and the variant character can be used to indicate control information such as halt identification number (see note 2).

WORD MARKS

Formats a, b, c, and d:

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	A_p	B_p
Format b:	JI (A)	A	NXT
Format c:	NXT	A	B
Format d:	NXT	A	B

NOTES

1. If a Halt instruction (in any format) is executed during a peripheral transfer, the transfer continues until it is completed.

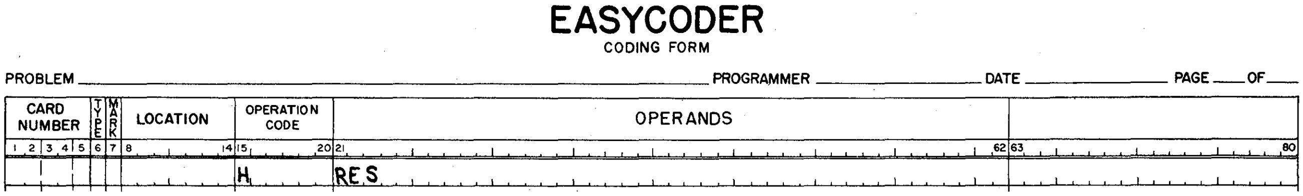
2. Formats c. and d. are useful when a program contains a number of halts. By assigning a number or symbol in the A and B addresses to each halt, the programmer can later identify a particular halt by displaying the contents of AAR and/or BAR. Although the contents of the variant register cannot be displayed through the console or control panel, format d. can be used to store a variant character which can subsequently be used by the program.

3. The Halt op code is a "privileged" op code that has special significance when the Type 1201, 1251, 2201, or 4201 central processor is equipped with the Storage Protect Feature (see Appendix E).

4. This instruction can be coded only in formats a., b., and c. when programming for the Type 201 or 201-1 processor.

EXAMPLES

1. Stop the machine and specify that when the RUN button is pressed, the next instruction will be selected from the location tagged RES.



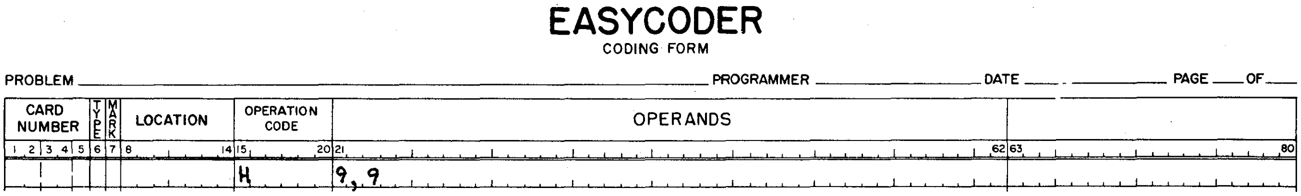
EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ___ OF ___

CARD NUMBER 1-5	TYPE 6	MARK 7	LOCATION 8-14	OPERATION CODE 15-20	OPERANDS 21-62	63-80
				H	RES	

2. Identify the halt at the end of a job as follows:

   A address =9
   B address =9



EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ___ OF ___

CARD NUMBER 1-5	TYPE 6	MARK 7	LOCATION 8-14	OPERATION CODE 15-20	OPERANDS 21-62	63-80
				H	9,9	

NOP | NO OPERATION

FORMAT

OP CODE	A ADDRESS	B ADDRESS
[redacted]		

FUNCTION

This instruction performs no operation. This op code can be substituted for the op code of any instruction to make that instruction ineffective.

WORD MARKS

Program operation resumes at the next op code identified by a word mark.

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR
NXT	A_p	B_p

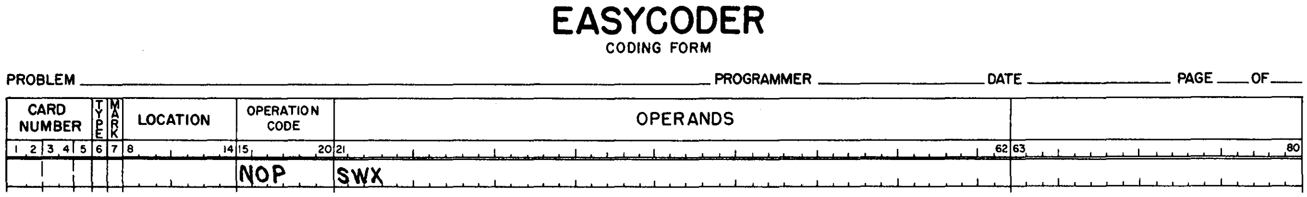
NOTES

1. This instruction is commonly used in program modification to cause the machine to skip over specific instructions.

2. Information appearing in an address portion of an instruction for which the NOP instruction is substituted is not loaded into the associated operand address register. The final character of such information, however, is loaded into the variant register.

EXAMPLE

Reserve the necessary storage locations for an instruction such as Branch (B/A) and substitute the op code NOP in this instruction. When the op code B is restored, the NOP instruction will be modified to branch to location SWX.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				NOP	SWX	

MCW MOVE CHARACTERS TO WORD MARK

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	■	■	■
b.	■	■	
c.	■		

FUNCTION

Format a: The data and item-mark bits in the A field are moved to the B field.

Format b: The data and item-mark bits in the A field are moved to the field specified by the contents of the B-address register (BAR).

Format c: The data and item-mark bits in the field specified by the contents of the A-address register (AAR) are moved to the field specified by the contents of BAR.

WORD MARKS

Formats a, b, and c:

A word mark (or record mark) is required in the shorter of the two fields. The operation terminates when this mark is sensed.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	A-N_w	B-N_w
Format b:	NXT	A-N_w	B_p-N_w
Format c:	NXT	A_p-N_w	B_p-N_w

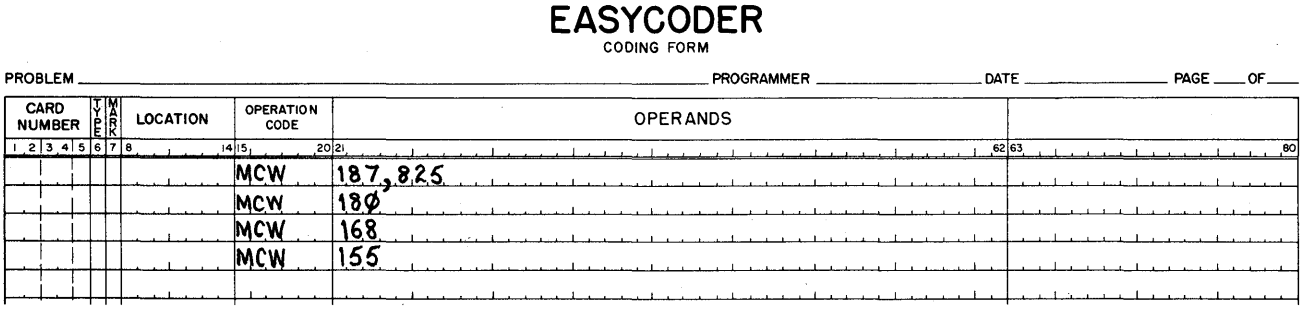
NOTE

Item marks initially stored in B-field locations will be cleared if the corresponding A-field characters do not include item marks.

EXAMPLE

Move the following A fields and store them in sequential B fields as shown.

Description	A field	B field
Unit Number	150-155	800-805
Rack Number	160-168	806-814
Part Number	173-180	815-822
Pin Number	185-187	823-825



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER 1 2 3 4 5	TYPE 6	MARK 7	LOCATION 8 14	OPERATION CODE 15 20	OPERANDS 21 62	63 80
1					MCW	187,825	
2					MCW	18Ø	
3					MCW	168	
4					MCW	155	
5							

LCA	LOAD CHARACTERS TO A-FIELD WORD MARK

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	■	■	■
b.	■	■	
c.	■		

FUNCTION

Format a: The data and punctuation bits in the A field are transferred to the B field.

Format b: The data and punctuation bits in the A field are transferred to the field specified by the contents of the B-address register (BAR).

Format c: The data and punctuation bits in the field specified by the contents of the A-address register (AAR) are transferred to the field specified by the contents of BAR.

WORD MARKS

Formats a, b, and c:

The A operand must have a defining word mark (or record mark). The operation terminates when this mark is transferred to the B field.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	$A\text{-}N_a$	$B\text{-}N_a$
Format b:	NXT	$A\text{-}N_a$	$B_p\text{-}N_a$
Format c:	NXT	$A_p\text{-}N_a$	$B_p\text{-}N_a$

NOTES

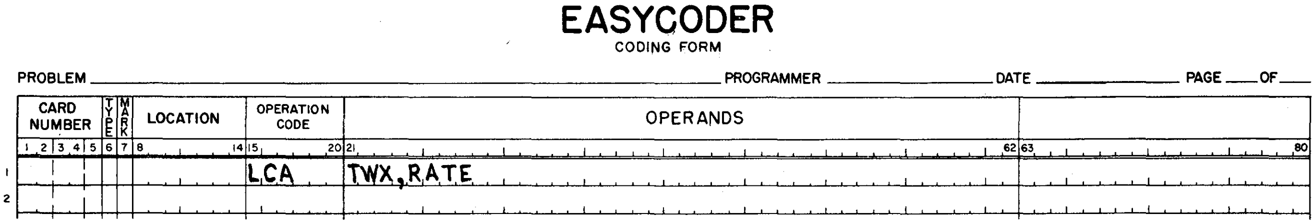
1. This instruction (in any format) is the only instruction that always moves both a field and its defining punctuation mark.

2. All punctuation (word marks, item marks, and record marks) initially stored in B-field locations will be cleared if the corresponding A-field characters do not include identical punctuation.

3. The B address must never fall within the A field. The A address may fall within the B field, however, if desired.

EXAMPLE

Move both the data bits and the defining word mark of the field tagged TWX to the field tagged RATE.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					LCA	TWX,RATE	
2							

SCR | STORE CONTROL REGISTERS

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.	■	■		■
b.	■	■		
c.	■			

FUNCTION

Format a: The contents of the control memory register specified by the variant character are stored in the field whose units position is defined by the A address of this instruction. The method of storing these contents depends on the addressing mode being used, as shown in Table 8-12. The valid variant characters and the control register each character represents are listed in Table 8-13.

Table 8-12. Control Register Contents Stored by SCR Instruction

Addressing Mode	Amount of Control Register Stored[1]
Two-Character	Low-order two characters (12 bits).
Three-Character	Low-order 15 bits; the high-order three bits of the field specified by the A address are cleared to zeros.
Four-Character	The entire contents of the control register plus sufficient high-order zeros to make up 18 bits (see page 2-7). For the Type 4201 processor, the entire 19 bits of the control register plus five high-order zeros.[2]

[1] All bit positions not required to address the largest memory address in a user's system are set to zeros in the A field.

[2] The five high-order bits of the high-order character are reset to zeros only in the 4201. In other processors, the entire six bits of the high-order character remain unchanged.

Format b: The contents of the control memory register specified by the variant character in a previous instruction are stored in the field whose units position is defined by the A address of this instruction. The number of bits stored depends on the addressing mode being used, as shown in Table 8-12. The valid variant characters and the control register each character represents are listed in Table 8-13.

Format c: The contents of the control memory register specified by the variant character in a previous instruction are stored in the field whose units position is defined by the contents of the A-address register (AAR). The number of bits stored depends on the addressing mode being used, as shown in Table 8-12. The valid variant characters and the control register each character represents are listed in Table 8-13.

Table 8-13. Control Registers Stored by SCR Instruction

Variant Character (Octal)	Control Register	Variant Character (Octal)	Control Register	Variant Character (Octal)	Control Register
00	CLC8	20	CLC9	64	CSR
01	CLC1	21	CLC4	66	EIR
02	CLC2	22	CLC5	67	AAR
03	CLC3	23	CLC6	70	BAR
04	CLC8'	24	CLC9'	76	IIR
05	CLC1'	25	CLC4'	77	SR
06	CLC2'	26	CLC5'		
07	CLC3'	27	CLC6'		
10	SLC8	30	SLC9		
11	SLC1	31	SLC4		
12	SLC2	32	SLC5		
13	SLC3	33	SLC6		
14	SLC8'	34	SLC9'		
15	SLC1'	35	SLC4'		
16	SLC2'	36	SLC5'		
17	SLC3'	37	SLC6'		

WORD MARKS

Formats a, b, and c:

A-operand punctuation neither affects nor is affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

Formats a, b, and c:

SR	AAR	BAR
NXT	A_p	B_p

NOTES

1. If AAR is specified by the variant character (octal 67), the previous address in AAR (not the A address retrieved from this instruction) is stored in the location specified by the A address.

2. The control memory register actually designated by the variant character 67_8 is a work register (not AAR). During the extraction of an SCR or LCR

instruction (see below), ARR is used to reference the main memory. Prior to this, the previous contents of AAR are stored in the work register; at the end of the instruction, the contents of the work register are restored in AAR.

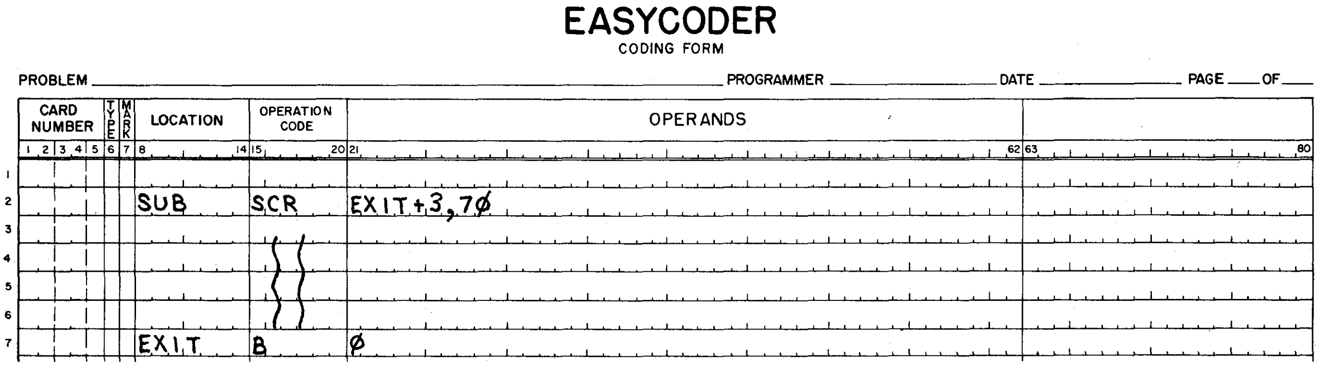
3. This instruction can be coded only in format a. when programming for the Type 201 or 201-1 processor.

4. In a processor equipped with the Scientific Unit (see Appendix F), this instruction must not be used to store the contents of the floating-point accumulators.

5. The SCR op code is a "privileged" op code that has special significance when the Type 1201, 1251, 2201, or 4201 central processor is equipped with the extended multiprogramming feature (see Appendix G).

EXAMPLE

Store the contents of BAR in the A address of the Branch instruction tagged EXIT. (The processor is assumed to be in the three-character addressing mode.)



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
1							
2				SUB	SCR	EXIT+3,7Ø	
3							
4							
5							
6							
7				EXIT	B	Ø	

LCR LOAD CONTROL REGISTERS FEATURE 011

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.	■	■		■
b.	■	■		
c.	■			

FUNCTION

Format a: The contents of the field specified by the A address are loaded into the control register specified by the variant character. The contents of the A field is another main memory address. The method of loading this address into the specified control register depends on the addressing mode being used, as shown in Table 8-14.

Table 8-14. Control Register Contents Loaded by LCR Instruction

Addressing Mode	Amount of Memory Address Loaded
Two-Character	Two-character (12 bit) address is loaded into the low-order two character locations of the register. All other bits in the register (if any) are not disturbed (i.e., the bank bits are protected).
Three-Character	15-bit address is loaded into the low-order 15-bit locations of the register. All other bits in the register (if any) are not disturbed (i.e., the sector bits are protected).
Four-Character	For processors other than the Type 4201, an address up to 18 bits long is loaded into the register; the number of bits loaded depends on the size of main memory (see Table 2-2, page 2-7). The Type 4201 processor always loads 19 bits. Thus, programs written for any other processor which are to be compatible with the Type 4201 must correctly set up the bit to the left of the stored 18-bit address before executing a 4-character LCR instruction.

Variant characters and their associated control registers are the same as those specified for the Store Control Registers instruction (see Table 8-13).

Format b: The contents of the field specified by the A address are loaded into the control register specified by the variant character in a previous instruction. The method of loading the contents of this field (another main memory address) depends on the addressing mode being used, as shown in Table 8-14. Variant characters and their associated control registers are the same as those specified for the Store Control Registers instruction.

Format c: The main memory address specified by the contents of the A-address register (AAR) is loaded into the control register specified in a previous instruction. The method of loading this address into the specified register depends on the addressing mode being used, as shown in Table 8-14. Variant characters and their associated control registers are the same as those specified for the Store Control Registers instruction.

WORD MARKS

Formats a, b, and c:

A-operand punctuation neither affects nor is affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

Formats a, b, and c:

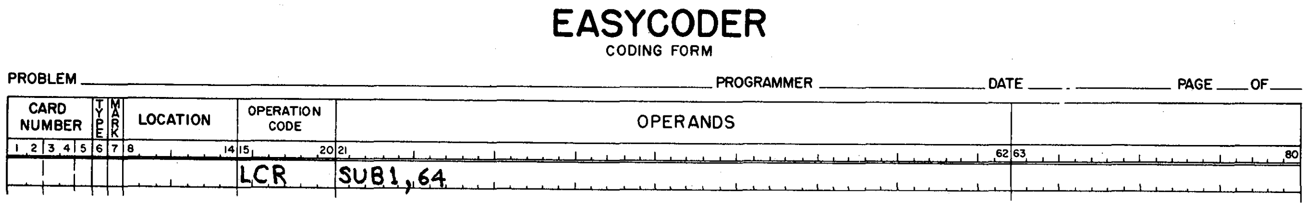
SR	AAR	BAR	
NXT	(A)	B_p	VARIANT = 67_8
NXT	A_p	(A)	VARIANT = 70_8
(A)	A_p	B_p	VARIANT = 77_8
NXT	A_p	B_p	ALL OTHERS

NOTES

1. If SR is specified by the variant character (77_8), the next instruction is selected from the location whose address is stored in the field specified by the A address of the Load Control Registers instruction. In all other cases, the program continues in sequence.
2. This instruction can be coded only in format a. when programming for the Type 201 or 201-1 processor.
3. The LCR op code is a "privileged" op code which has special significance when used with a Type 1201, 1251, 2201, or 4201 processor equipped with the Storage Protect Feature (see Appendix E).
4. In a processor equipped with the Scientific Unit (see Appendix F), this instruction must not be used to load the floating-point accumulator.

EXAMPLE

Load the address stored in the location tagged SUB1 into the change sequence register (CSR).



EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ____ OF ____

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				LCR	SUB1,64	

CAM | CHANGE ADDRESSING MODE | FEATURE 011[1]

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.	■			■
b.	■			

FUNCTION

Format a: The Change Addressing Mode instruction is used to specify the following conditions, as designated by the variant character:

[1] The CAM instruction is included in Type 201 and 201-1 processors whose memory capacities exceed 4,096 characters, whether or not Feature 011 is present.

1. The addressing mode (two-, three-, or four-character) in which the processor is to interpret the address portions of all subsequent instructions (see note 1).
2. The processing mode (standard mode or "trap" mode) in which all subsequent instructions are to be processed. (See note 3 for a description of the trap mode.)
3. The "S" mode of processing in which several Series 200 instructions are defined in a special manner (see note 4).

The variant characters and the mode(s) each character represents are listed in Table 8-15.

Format b: The variant character in a previous instruction specifies the addressing mode and processing mode in which all subsequent instructions are to be processed. The variant characters and the mode(s) each character represents are listed in Table 8-15.

Table 8-15. Modes Specified by Variant Character in CAM Instruction

Variant Character (Octal)	Mode(s)
20	Two-character, standard mode
00 or 40	Three-character, standard mode
60	Four-character, standard mode
24	Two-character, trap mode
04 or 44	Three-character, trap mode
64	Four-character, trap mode
30	Two-character, "S" mode
10	Three-character, "S" mode
70	Four-character, "S" mode
34	Two-character, trap, "S" mode
14	Three-character, trap, "S" mode
74	Four-character, trap, "S" mode

WORD MARKS

Formats a and b:

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

Formats a and b:

SR	AAR	BAR
NXT	A_p	B_p

NOTES

1. The CAM instruction is used in conjunction with the ADMODE assembly control statement to specify addressing mode. (See page 7-11 for a description of the ADMODE statement.) The ADMODE statement directs the Assembly Program to assemble the address portions of all subsequent source program instructions as two-, three-, or four-character addresses. The CAM instruction directs the processor to interpret the address portions

of all subsequent object program instructions as two-, three-, or four-character addresses. Thus, an address assembled in the three-character addressing mode (via an ADMODE statement) must be processed during a program run as a three-character address for proper execution; the processor is placed in the three-character addressing mode during object program execution by the CAM instruction.

2. The ability to change addressing modes within a program makes it possible to save both time and memory space and provides greater programming flexibility. Extraction and execution time is saved when a smaller addressing mode is used, due to the elimination of the extra memory cycles necessary for a larger address (in characters). Memory space may be conserved by storing frequently used subroutines in the two-character addressing mode (see example).

   The larger addresses are necessary to address larger continuous portions of memory. For instance, a two-character address can specify only memory locations within a 4, 096-character bank of main memory. A three-character address can refer to any location in a 32, 768-character sector. A four-character address can directly address any location in the entire memory (from location 0_{10} to location $524, 288_{10}$).

3. The "trap" mode of instruction execution is included as part of Feature 010 for Type 201-2 processors, even though the CAM instruction is standard. In the Type 201 and 201-1 processors, both the CAM instruction and trap mode are included as part of Feature 011. When the processor is in the trap mode of instruction execution, any instruction whose op code contains an item mark (or record mark) is both extracted and executed as if it were a Change Sequencing Mode instruction (see page 8-66), regardless of the op code that is actually present. The A address, B address, and variant character (if any) of the instruction are delivered to AAR, BAR, and the variant register, respectively. The "trapped" op code is not executed; a Change Sequencing Mode instruction (CSM) is executed in its place. The CSM instruction causes a branch to the location stored in the change sequence register (CSR); this location is the beginning of a routine to interpret and execute the instruction whose op code was trapped.

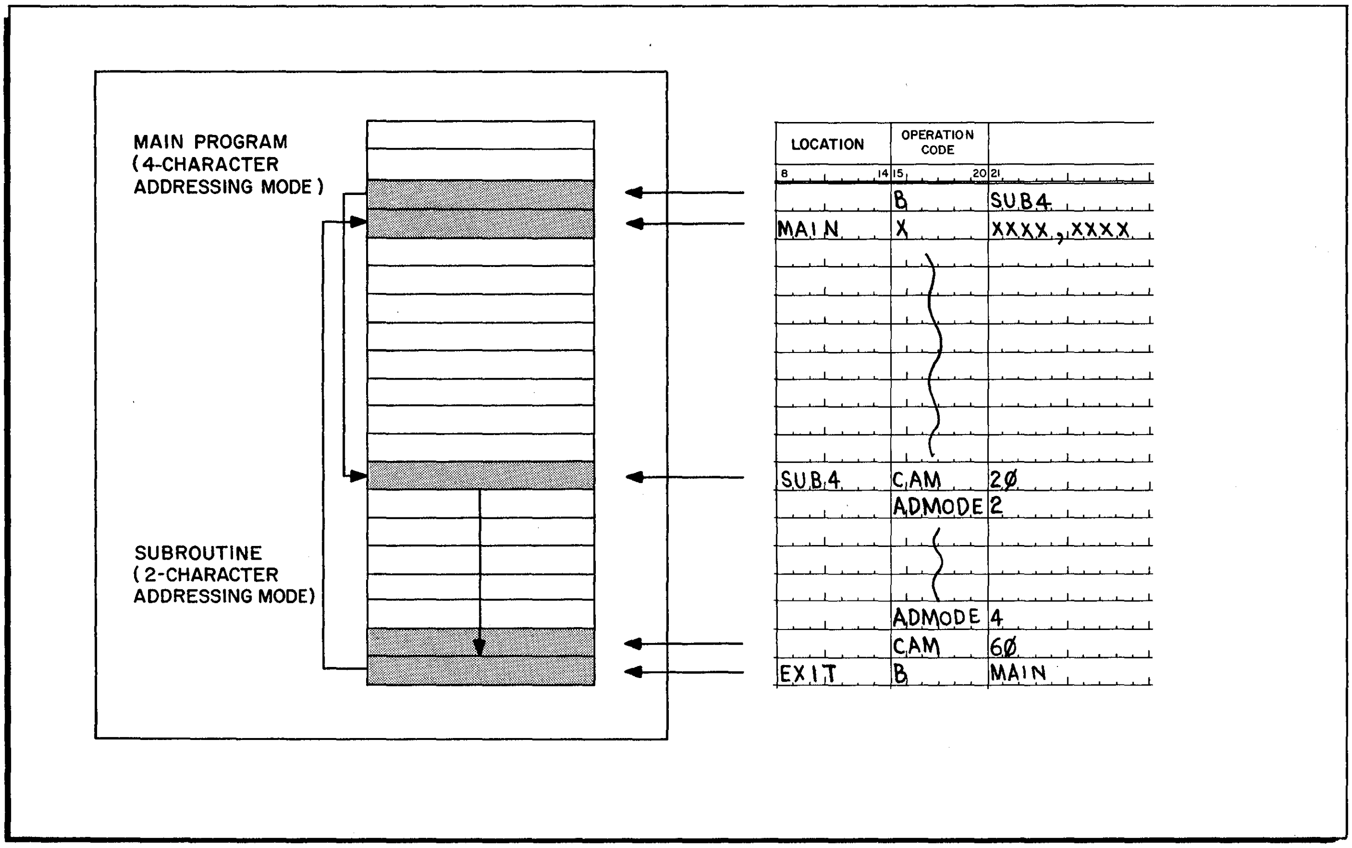
   The trap mode is used effectively by the Liberator conversion programs (Bridge and Easytran) to replace the seldom used instructions of competitive systems when converting the programs of these systems to Series 200 language. Such instructions are replaced by routines when the trapped op codes are executed as CSM op codes.

4. In addition to specify the standard or trap modes, the CAM instruction is used to specify the "S" mode of processing when Feature 0191 is included in a 1201, 2201, or 4201 central processor. When the processor is in the "S" mode the A, S, ZA, ZS, and BCE instructions are implemented in a special manner. The particular differences that result from the "S" mode of processing are indicated in the notes for that instruction.

5. This instruction can be coded only in format a. when programming for the Type 201 or 201-1 processor.

EXAMPLE

Figure 8-5 shows the coding which provides entry to and exit from a subroutine to be executed in the two-character addressing mode. Both an ADMODE statement and a CAM instruction must be coded (in either order) at the beginning and end of the subroutine. However, only the CAM instructions are stored in the main memory. (Since CAM instructions have no address portions, the manner in which they are stored is not affected by an ADMODE statement.)



LOCATION (8–14)	OPERATION CODE (15–20)	(21)
	B	SUB4
MAIN	X	XXXX,XXXX
SUB4	CAM	2Ø
	ADMODE	2
	ADMODE	4
	CAM	6Ø
EXIT	B	MAIN

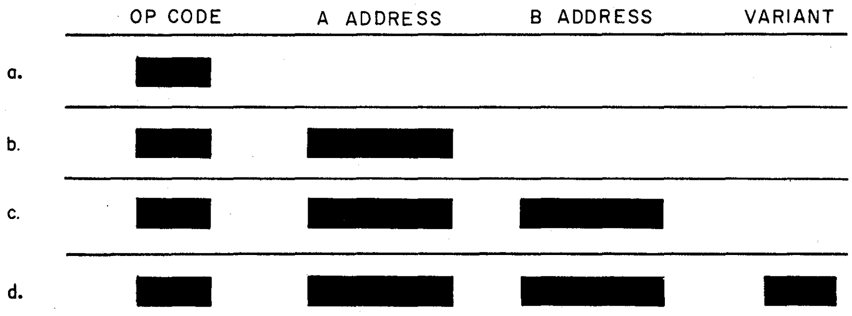
Figure 8-5. Changing Addressing Modes via CAM Instruction

NOTE: The branch from the main program to SUB4 in Figure 8-5 could have been caused by an item-marked op code (if the processor were in the trap mode) instead of by the Branch instruction. In this case, the memory location tagged SUB4 would be stored in CSR, so that when the item-marked op code was encountered, the contents of SR and CSR would be interchanged. The program would automatically branch to SUB4 in this case.

CSM	CHANGE SEQUENCING MODE

FEATURES 010 & 011

FORMAT



FUNCTION

Format a: The contents of the sequence register (SR) and the change sequence register (CSR) are interchanged, and the program branches to the address which was previously stored in CSR.

Format b: The contents of SR and CSR are interchanged, and the program branches to the address which was previously stored in CSR. The A address is loaded into the A-address register (AAR).

Format c: The contents of SR and CSR are interchanged, and the program branches to the address which was previously stored in CSR. The A and B addresses are loaded into AAR and BAR, respectively.

Format d: The contents of SR and CSR are interchanged, and the program branches to the address which was previously stored in CSR. The A and B addresses and the variant character are loaded into AAR, BAR, and the variant register, respectively.

WORD MARKS

Formats a, b, c, and d:

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

	SR	CSR	AAR	BAR
Format a:	JI (contents of CSR)	NXT	A_p	B_p
Format b:	JI (contents of CSR)	NXT	A	B_p
Format c:	JI (contents of CSR)	NXT	A	B
Format d:	JI (contents of CSR)	NXT	A	B

NOTES

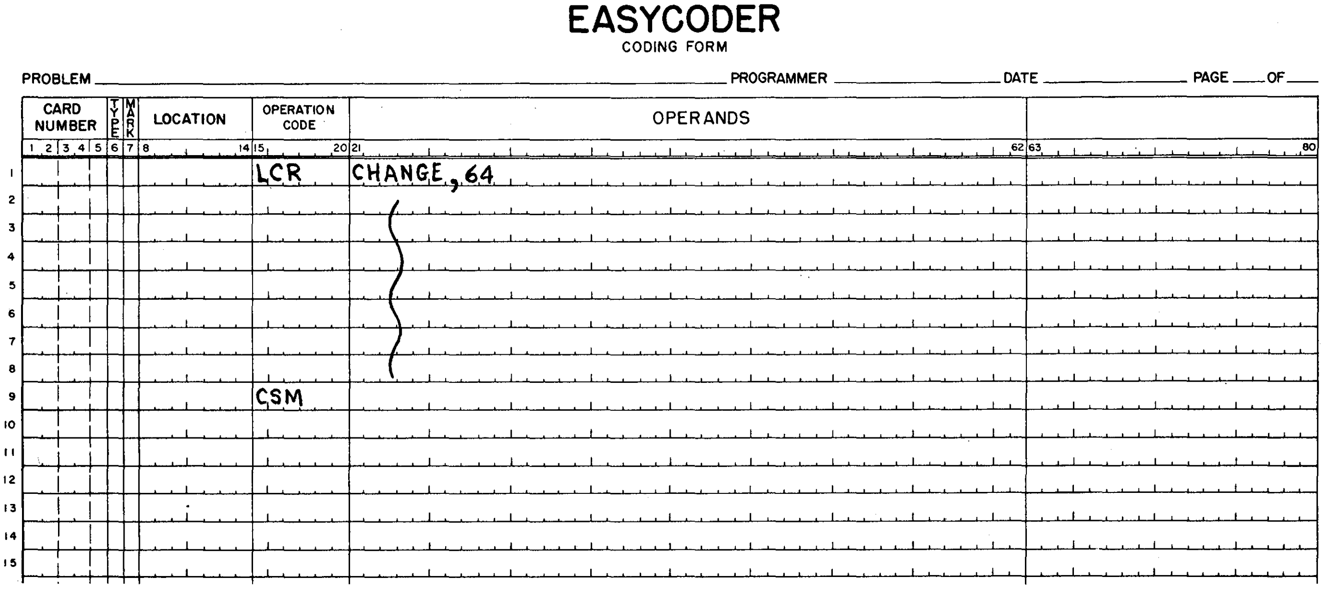
1. The Load Control Registers instruction (see page 8-60) can be used to set up the contents of CSR.

2. When the "trap" mode of instruction execution is specified by the Change Addressing Mode instruction (see page 8-62), any subsequent instruction whose op code contains an item mark or a record mark is retrieved and executed as if it were a Change Sequencing Mode instruction. An instruction which is "trapped" in this manner must conform to one of the valid formats of the CSM instruction.

3. This instruction can be coded only in formats a., b., and c. when programming for the Type 201 or 201-1 processor.

EXAMPLE

Store the absolute address tagged CHANGE in CSR via a Load Control Registers instruction. Later, alter the program sequence by branching to the instruction tagged CHANGE. Provide for the ultimate return to normal programming sequence by storing the contents of SR in CSR.



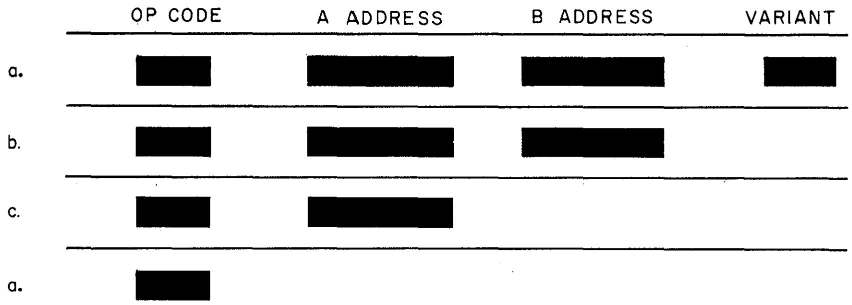
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
1					LCR	CHANGE,64	
2							
3							
4							
5							
6							
7							
8							
9					CSM		
10							
11							
12							
13							
14							
15							

EXM | EXTENDED MOVE | FEATURES 010 & 011

FORMAT



FUNCTION

Format a: The contents of the A field are moved to the B field in the manner specified by the variant character (see Table 8-16). The programmer specifies how the move operation is to be performed by selecting the desired conditions from the table and encoding the resulting two octal digits as the variant character of the instruction.

Format b: The contents of the A field are moved to the B field in the manner specified by the variant character of a previous instruction (see Table 8-16).

Format c: The contents of the A field are moved to the field specified by the contents of the B-address register (BAR) in the manner specified by the variant character of a previous instruction (see Table 8-16).

Format d: The contents of the field specified by the contents of the A-address register (AAR) are moved to the field specified by the contents of BAR in the manner specified by the variant character of a previous instruction (see Table 8-16).

Table 8-16. Extended Move Conditions

Variant Character (Octal)	Condition
X1	Move A-field data bits to corresponding bit positions in B field.
X2	Move A-field word-mark bits to corresponding bit positions in B field.
X3	Move A-field data and word-mark bits to corresponding bit positions in B field.
X4	Move A-field item-mark bits to corresponding bit positions in B field.
X5	Move A-field data and item-mark bits to corresponding bit positions in B field.
X6	Move A-field word-mark and item-mark bits to corresponding bit positions in B field.
X7	Move A-field data, word-mark and item-mark bits to corresponding bit positions in B field.
0X	Move one character from A to B. The A- and B-address registers are decremented by one.
1X	Move one character from A to B. The A- and B-address registers are incremented by one.
2X	Move characters from right to left (A and B addresses specify rightmost characters in operand fields). Terminate the operation when the first A-field word mark is sensed.
3X	Move characters from left to right (A and B addresses specify leftmost characters in operand fields). Terminate the operation when the first A-field word mark is sensed.

Table 8-16 (cont). Extended Move Conditions

Variant Character (Octal)	Condition
4X	Move characters from right to left. Terminate the operation when the first A-field item mark is sensed.
5X	Move characters from left to right. Terminate the operation when the first A-field item mark is sensed.
6X	Move characters from right to left. Terminate the operation when the first A-field record mark is sensed.
7X	Move characters from left to right. Terminate the operation when the first A-field record mark is sensed.

PUNCTUATION MARKS

Formats a, b, c, and d:

The A field must have a defining punctuation mark, except when the variant character specifies a one-character transfer.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR	
Format a:	NXT	$A-N_a$	$B-N_a$	VARIANT = (0, 2, 4, or 6)X
	NXT	$A+N_a$	$B+N_a$	VARIANT = (1, 3, 5, or 7)X
Format b:	NXT	$A-N_a$	$B-N_a$	VARIANT = (0, 2, 4, or 6)X
	NXT	$A+N_a$	$B+N_a$	VARIANT = (1, 3, 5, or 7)X
Format c:	NXT	$A-N_a$	B_p-N_a	VARIANT = (0, 2, 4, or 6)X
	NXT	$A+N_a$	B_p+N_a	VARIANT = (1, 3, 5, or 7)X
Format d:	NXT	A_p-N_a	B_p-N_a	VARIANT = (0, 2, 4, or 6)X
	NXT	A_p+N_a	B_p+N_a	VARIANT = (1, 3, 5, or 7)X

NOTES

1. This instruction can be coded only in formats a. and d. when programming for the Type 201 or 201-1 processor.

2. Here is an example of a typical variant bit configuration: V = 110011. This configuration, encoded in octal notation as 63, specifies that A-field data and word-mark bits are to be moved to the B field from right to left until the first record mark is sensed in the A field.

3. Consider the variant character in its six-bit form, $V_6V_5V_4V_3V_2V_1$. If $V_1 = 0$, A-operand data bits are not transferred and data bits in the B field remain unchanged.

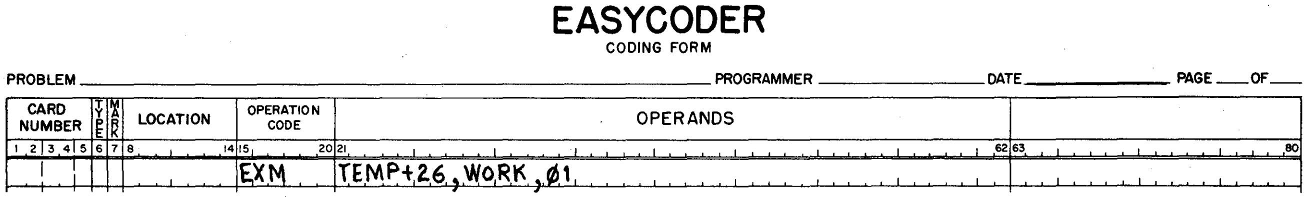
4. If $V_2 = 0$, A-operand word-mark bits are not transferred and B-operand word-mark bits remain unchanged.

5. If $V_3 = 0$, A-operand item-mark bits are not transferred and B-operand item-mark bits remain unchanged.

6. The character containing the terminating punctuation is moved in the same manner as the rest of the field.

EXAMPLES

1. Move the data bits of the single character in the location 26 beyond that tagged TEMP to the location tagged WORK, and decrement the A- and B-address registers.

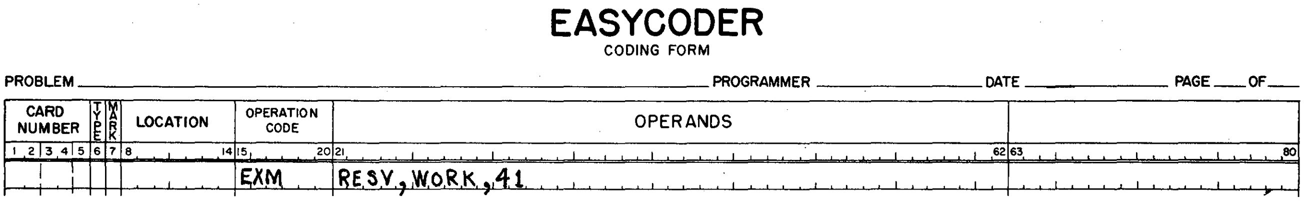


EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				EXM	TEMP+26,WORK,Ø1	

2. Move only the data bits in the field tagged RESV to the field tagged WORK. Move the data from right to left, and terminate the operation when the first item mark in the A field is sensed.



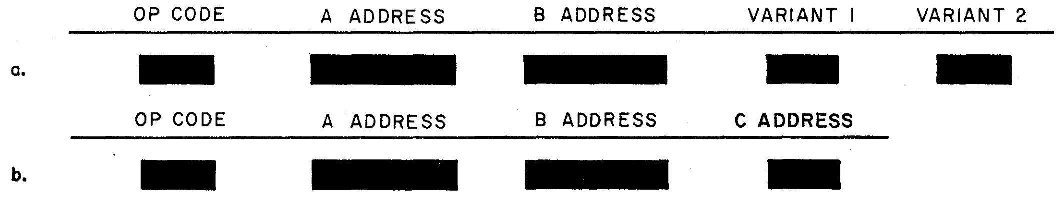
EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				EXM	RESV,WORK,41	

MAT | MOVE AND TRANSLATE | FEATURES 010 & 011

FORMAT



FUNCTION

Format a: The MAT instruction translates characters from one six-bit configuration to another by means of a stored "translation table." The instruction can be used to translate any number of consecutive characters in the memory.

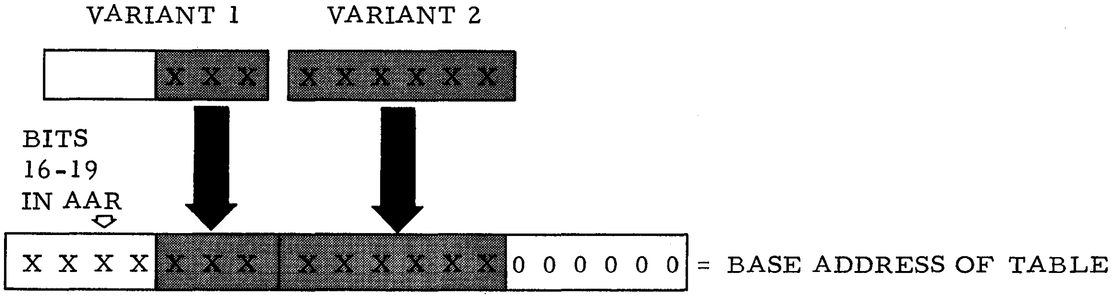
The A address specifies the location of the rightmost character in the field to be translated. The B address specifies the location into which the translated equivalent of the rightmost A-field character will be moved.

The operation normally terminates when an A-field word mark is sensed. The operation is also terminated if a character is transferred from a word-marked location within the translation table.

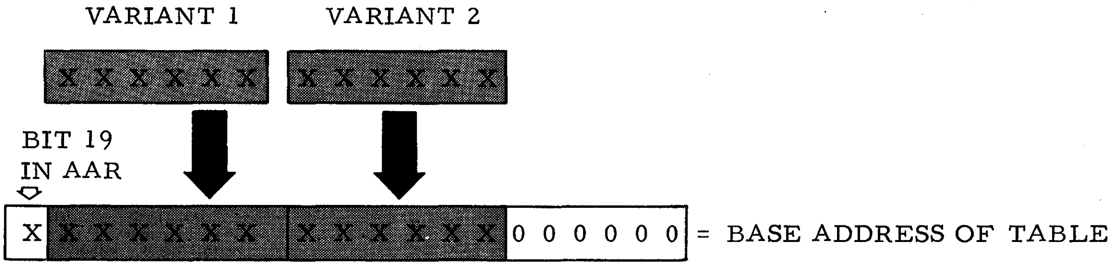
The address within the translation table which contains the translated equivalent of an A-field character is formed by combining the A-field character with the two variant characters. The method of combining these three characters depends on the addressing mode being used, as described below.

The leftmost, or base, address of the translation table is formed by combining variants 1, 2, and a zero character as shown below. If the processor is in the two- or three-character addressing mode, the leftmost three bits of variant 1 are ignored and the corresponding bit positions (i.e., the sector bits) in the base address (bits 16, 17, 18 and 19) are taken from the contents of the A-address register (AAR). If the processor is in the four-character addressing mode (see below), the entire six-bit contents of variant 1 form bits 13-18 of the base address, while the leftmost (nineteenth) bit, if present, is taken from the contents of AAR.

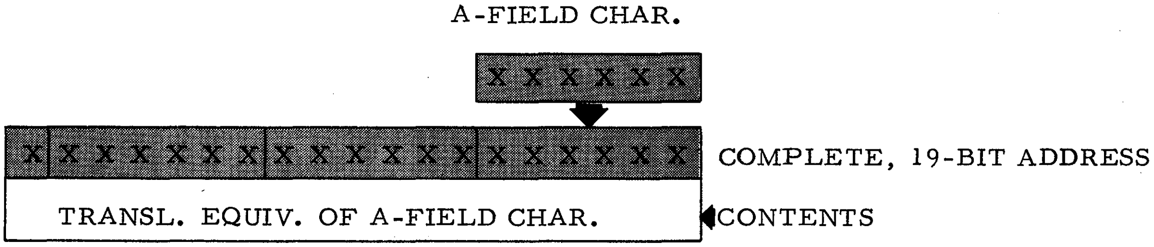
Two- or Three-Character Addressing Mode



Four-Character Addressing Mode



A character in the A field is translated when it is appended to the variant characters (in place of the zero character) to form a complete, 19-bit address. This complete address contains the translated equivalent of the appended A-field character (see below).



Note that because of the positions of variant 1 and variant 2 in the complete address, the base address of the table will always be a multiple of 64. This is compatible with translation requirements since each A-field character can have any of 64 bit configurations (see note 5).

It is a simple task to store the desired equivalent values in a translation table. For instance, assume that a character set which is to be translated into Honeywell code represents the letter A by the bit configuration 110001. Since this bit configuration represents a binary value of 49, the desired Honeywell equivalent (i.e., 010001) should be stored 49 locations beyond the base address of the translation table.

Format b: This is an alternate and simplcr format for coding the MAT instruction. In this format, a "C address" replaces the variant characters used in format a. to define the base address of the table. Thus, format b. relieves the programmer of dealing with modulo-64 addresses and converting to octal notation each time a MAT instruction is coded.

The C address is a symbolic tag that is contained in the location field of another source-program entry (e.g., a RESV statement). Once the absolute base address of the table is defined as described for format a., the C address is equated to that address and used in its stead whenever a MAT instruction using the same table is coded again in the program.

Example 2 shows how a C address can be used to define the base address of the translation table.

WORD MARKS

Formats a and b:

The A field must have a defining word mark. It is this word mark that normally stops the operation. The operation will also be terminated if a character is transferred from a word-marked location within the translation table.

ADDRESS REGISTERS AFTER OPERATION

Formats a and b:

SR	AAR	BAR
NXT	A-N_{ct}	B-N_{ct}

NOTES

1. This instruction cannot be chained.
2. The contents of the variant register following a move and translate operation are unspecified. Therefore, an instruction requiring a variant character must not be chained after an MAT instruction.
3. Item-mark bits as well as data bits are transferred from the translation table to the B field.
4. Word marks initially stored in the B field remain unchanged. They do not affect the execution of this instruction.
5. The base address of the translation table must always be a multiple of 64. The Easycoder Assembly Program automatically stores the table in this manner when directed by a MORG assembly control statement (see page 7-9) containing an operand of 64.

EXAMPLES

1. Figure 8-6 shows how A-field data is moved to the B field via a translation table.

   Translate the contents of the field tagged EXCODE using the stored translation table whose base address is 256_{10} (=400_8). Store the translated equivalent in the field tagged EQUIV.

   A Address: EXCODE (absolute value = location 630)

   B Address: EQUIV (absolute value = location 900)

   Variant 1: 00 =

   Variant 2: 04 = base address of table (location 256)

EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				MAT	EXCODE,EQUIV,ØØ,Ø4	

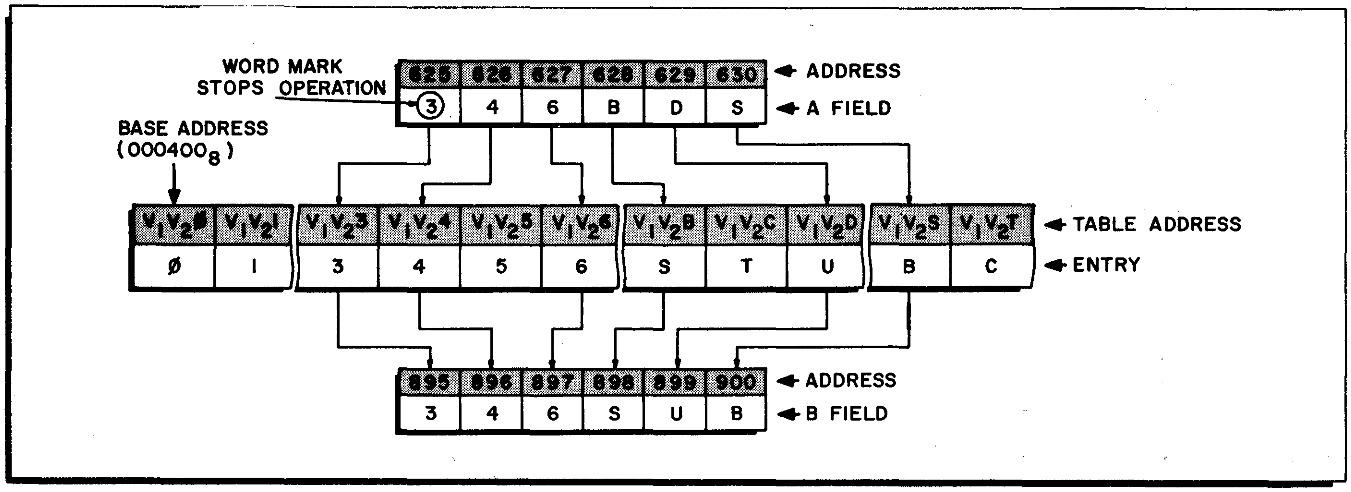

WORD MARK STOPS OPERATION
BASE ADDRESS (000400_8)
625 626 627 628 629 630
ADDRESS
3 4 6 B D S
A FIELD
$V_1V_2Ø$ V_1V_21 V_1V_23 V_1V_24 V_1V_25 V_1V_26 V_1V_2B V_1V_2C V_1V_2D V_1V_2S V_1V_2T
TABLE ADDRESS
Ø 1 3 4 5 6 S T U B C
ENTRY
895 896 897 898 899 900
ADDRESS
3 4 6 S U B
B FIELD

Figure 8-6. MAT Operation

2. The following coding shows how the preceding MAT instruction can be coded using a C address. The translation table is set up with a base address of 256_{10} by means of an ORG statement and two DC statements. The ORG statement directs the Assembly Program to load subsequent coding into memory locations beginning at location 256_{10}. The first DC statement defines an alphanumeric constant 40 characters long (i.e., the maximum size of an alphanumeric constant). These characters are the first 40 characters of a 64-character translation table. The second DC statement defines the remaining 24 characters of the table.

   When the MAT instruction is executed, the absolute address equated to the tag MATAB1 is used as the table's base address as in example 1.

EASYCODER

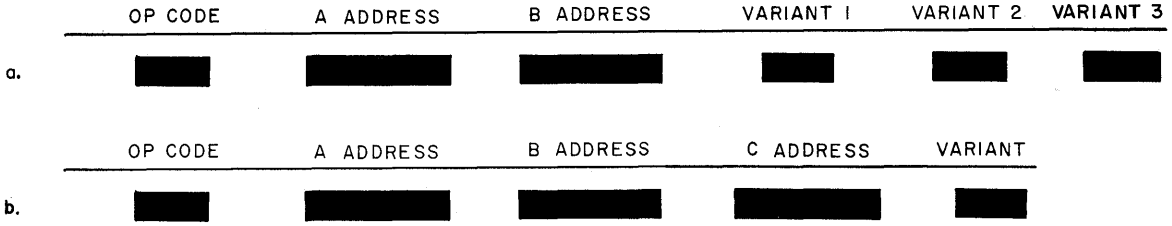
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					QRG	256	
2				MATAB1	DC	*Δ1234567890̸$\substack{X\\8\\2}\substack{Ø\\8\\2}$@$\substack{8\\6}$Δ$\substack{Ø\\8\\7}$/STUVWXYZΔ,$\substack{R\\8\\3}$#$\substack{8\\2}\substack{X\\8\\5}$-JKLMNOP*	
3					DC	@QRō$*□$\substack{Ø\\8\\2}$Δ$\substack{8\\7}$ABCDEFGHIΔ.$\substack{Ø\\8\\6}$%$\substack{8\\5}$Δ@	
4							
5							
6							
7							
8					MAT	EXCODE,EQUIV,MATAB1	

MIT MOVE ITEM AND TRANSLATE

FORMAT



FUNCTION

Format a: The Move Item and Translate instruction is used to translate any information unit (up to 12-bit code) to another information unit of up to 12 bits (e.g., to Series 200 six-bit character code) by the use of a stored translation table. Any number of consecutive information units stored in the memory can be translated.

The A address is the leftmost address of the item to be translated. The B address is the leftmost address of the item into which the translated equivalent of the A item will be moved. The MIT instruction translates the data contents in the A item and moves the translated results, left to right, to the B item.

The operation normally terminates when an item mark is sensed in the A item. The operation will also be terminated if a word-marked character is encountered in the translation table.

An information unit up to six bits in length is stored in one six-bit character location in the memory. Any information unit greater than six bits (7 through 12 bits) is stored in two successive six-bit character locations. Thus, an information unit consisting of up to six bits is considered as a six-bit character, and a unit of from 7 to 12 bits is considered as a "12-bit character."

The sizes of the information units involved in the operation are specified by variant 3, as shown in Table 8-17.

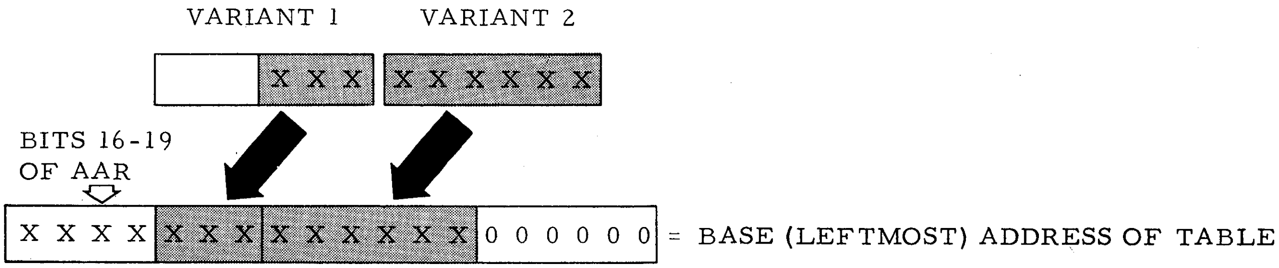
Table 8-17. Size of Information Units in MIT Operation

VARIANT 3	OPERATION
00	Translate each six-bit character in the A item. Move the translated equivalent to a six-bit character location in the B item.
01	Translate each 12-bit character in the A item. Move the translated equivalent to a six-bit character location in the B item.
02	Translate each six-bit character in the A item. Move the translated equivalent to two character locations (12 bits) in the B item.
03	Translate each 12-bit character in the A item. Move the translated equivalent to two character locations (12 bits) in the B item.

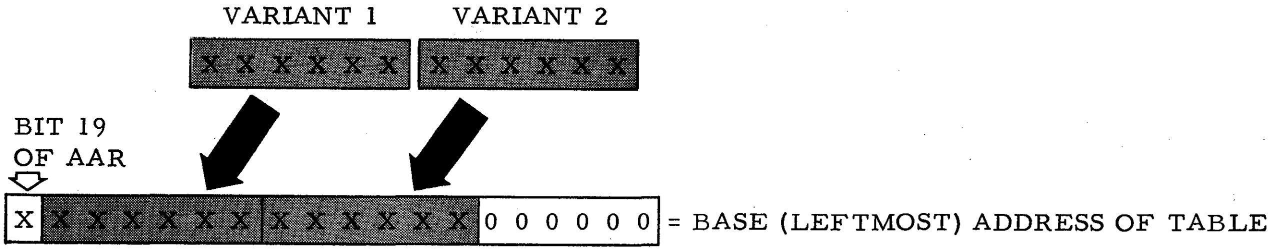
The desired equivalent of an A-item information unit is taken from the stored translation table and moved to the B item. Thus, if the desired equivalent is a six-bit character, each table entry occupies one six-bit character location in the table. If the desired equivalent is a 12-bit character, each table entry occupies two consecutive six-bit character locations in the table. Consequently, variant 3 implicitly specifies the size of each table entry when it explicitly specifies the size of the B-item information unit.

The leftmost, or base, address of the translation table is formed by combining variants 1, 2, and a zero character as shown below. If the processor is in the two- or three-character addressing mode, the leftmost three bits of variant 1 are ignored and the corresponding bit positions (i.e., the sector bits) in the base address of the table are taken from the contents of the A-address register (AAR). If the processor is in the four-character addressing mode, the entire six-bit contents of variant 1 form bits 13-18 of the base address, and the nineteenth bit, if present, is taken from the contents of AAR.

Two- or Three-Character Addressing Mode

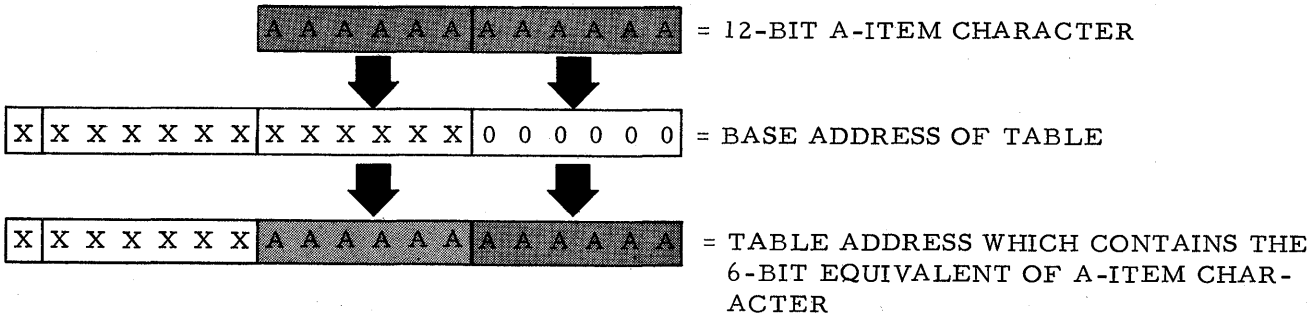


Four-Character Addressing Mode



The address within the translation table which contains the translated equivalent of an A-item character (6- or 12-bit) is formed by superimposing the A-item character over the base address of the table. The method of superposition depends on the size of each table entry (whether 6 or 12 bits), as described below.[1]

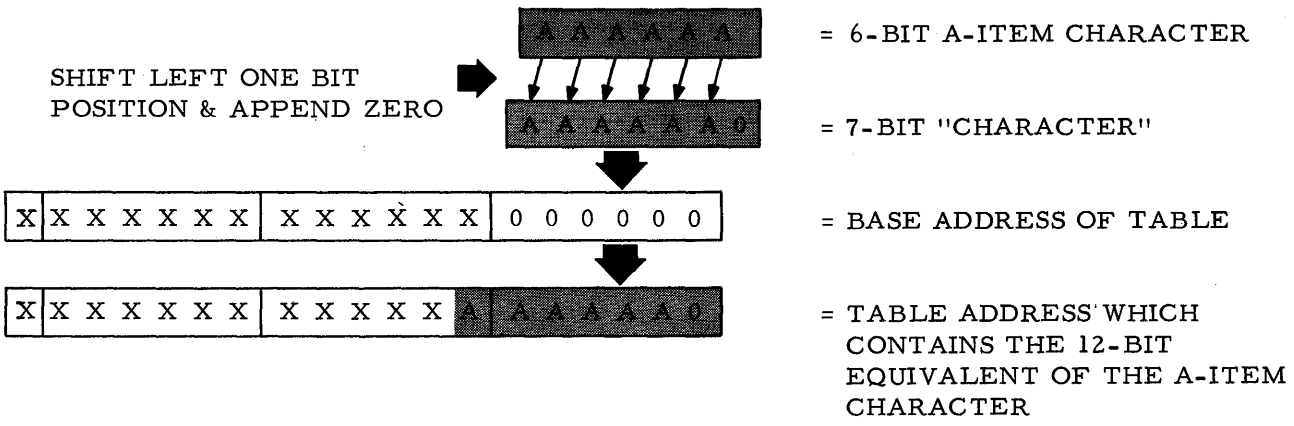
If each table entry is a six-bit character (variant 3 = 00 or 01), the 6- or 12-bit A-item character is superimposed over the rightmost bit positions of the base address. The illustration below shows a 12-bit A-item character being superimposed over the base address, where A = an A-item bit and X = a base address bit.



If each table entry is a 12-bit character (variant 3 = 02 or 03), the 6- or 12-bit A-item character is first shifted one bit position to the left, forming a 7- or 13-bit "character." The rightmost bit position of the character is set to zero. The "character" is then superimposed over the base address to form the table address of the translated equivalent of the A-item character. The shift operation is used to double the referenced table address, since each table entry is stored in two, rather than one, six-bit character locations. The resultant address is the address of the leftmost of the two successive six-bit character locations in the table.

The illustration below shows how a 6-bit A-item character is shifted one bit position to the left and then superimposed over the translation table's base address to form the table address of its equivalent; A = an A-item bit, and X = a base address bit.

[1] Superposition is performed by placing a 1 bit in every position of the table address in which a 1 existed in either the A-item character or the base address or both. This is the "logical inclusive OR" function.


A A A A A A
= 6-BIT A-ITEM CHARACTER
SHIFT LEFT ONE BIT POSITION & APPEND ZERO
A A A A A A 0
= 7-BIT "CHARACTER"
X X X X X X X X X X X X X X 0 0 0 0 0 0
= BASE ADDRESS OF TABLE
X X X X X X X X X X X X A A A A A A 0
= TABLE ADDRESS WHICH CONTAINS THE 12-BIT EQUIVALENT OF THE A-ITEM CHARACTER

<u>Format b</u>: This is an alternate format for coding the MIT instruction. As in the MAT instruction (see page 8-70), a symbolic tag replaces the variant characters used to define the base address of the table in format a. The tag is contained in the location field of another source-program entry which equates the tag to the base address of the table.

The second example of coding an MAT instruction (page 8-73) shows the method by which a translation table is stored in memory so that the leftmost location of the table can be used as a symbolic address. This is identical to the method used for format b. of the MIT instruction.

PUNCTUATION MARKS

Formats a and b:

The A item must contain an item mark. It is this punctuation mark that normally stops the operation. If the A-item information units are 12-bit characters, the terminating item mark may appear in either of the two six-bit character locations.

The operation will also be terminated if a character (6- or 12-bit) is encountered in a word-marked location in the translation table. In this case, neither the word-marked character nor any subsequent characters are moved to the B item; instead, a sequence change is performed (see note 5).

ADDRESS REGISTERS AFTER OPERATION

Formats a and b:

SR	CSR	AAR	BAR	
NXT	CSR_p	$A+(NA_u)(N_{ut})$	$B+(NB_u)(N_{ut})$	ITEM MARK IN A ITEM STOPS OPERATION
JI (contents of CSR)	NXT	$A+(NA_u)(N_{ut})$	$B+(NB_u)(N_{ut})$	WORD MARK IN TABLE STOPS OPERATION

NOTES

1. This instruction cannot be chained.
2. The last six-bit character referenced in the translation table (whether word-marked or not) is left in the variant register following the move item and translate operation.

3. Item-mark bits as well as data bits are transferred from the translation table to the B item.

4. Word marks initially stored in the B item remain unchanged. They do not affect the execution of this instruction.

5. A data control character (e.g., a case-shift character in a teletype code), rather than a translated equivalent to be transferred to the B item, can be stored in a word-marked location in the table. When this word-marked location is sensed, the character in that location is not moved; rather, the contents of SR and CSR are interchanged, providing entry to the routine whose beginning address was previously stored in CSR. Since the word-marked character is stored in the variant register (see note 2), that character can be stored by a Store Variant and Indicators instruction (see page 8-92) and subsequently tested for identification in the routine.

6. The base address of the translation table must be a multiple of at least 64, due to the positions of variants 1 and 2 in the total 19-bit address. This requirement is sufficient only for the translation of 6-bit to 6-bit codes. If other than 6-bit codes are involved in the translation, the base address of the table must be a multiple of X (where X is the product of the number of codes defined by active bits in the A field entries times the number of characters in each table entry). In other words, the base address of the table must be a multiple of the table size itself. The MORG assembly control statement (see page 7-9) can be used to assign memory locations to the translation table, starting with the next available memory location whose address is a multiple of 64, 128, 256, etc., as determined by the size of the table.

7. This instruction is a standard feature on all processors except the Types 201 and 201-1, on which it is not available.

EXAMPLE

Figure 8-7 shows how eight-bit code is translated to Series 200 six-bit character code by means of a stored translation table. Each eight-bit information unit is stored in two consecutive six-bit character locations in the A item tagged EIGHT.

Translate the data contents of the item tagged EIGHT using the translation table whose base address is location 512_{10} (1000_8). Store the translated values (six-bit characters) in the item tagged SIX.

A Address:	EIGHT (absolute value = location 800)
B Address:	SIX (absolute value = location 650)
Variant 1:	00 = } the address of table (location 512)
Variant 2:	10 = }
Variant 3:	01

EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
				MIT	EIGHT,SIX,ØØ,1Ø,Ø1	

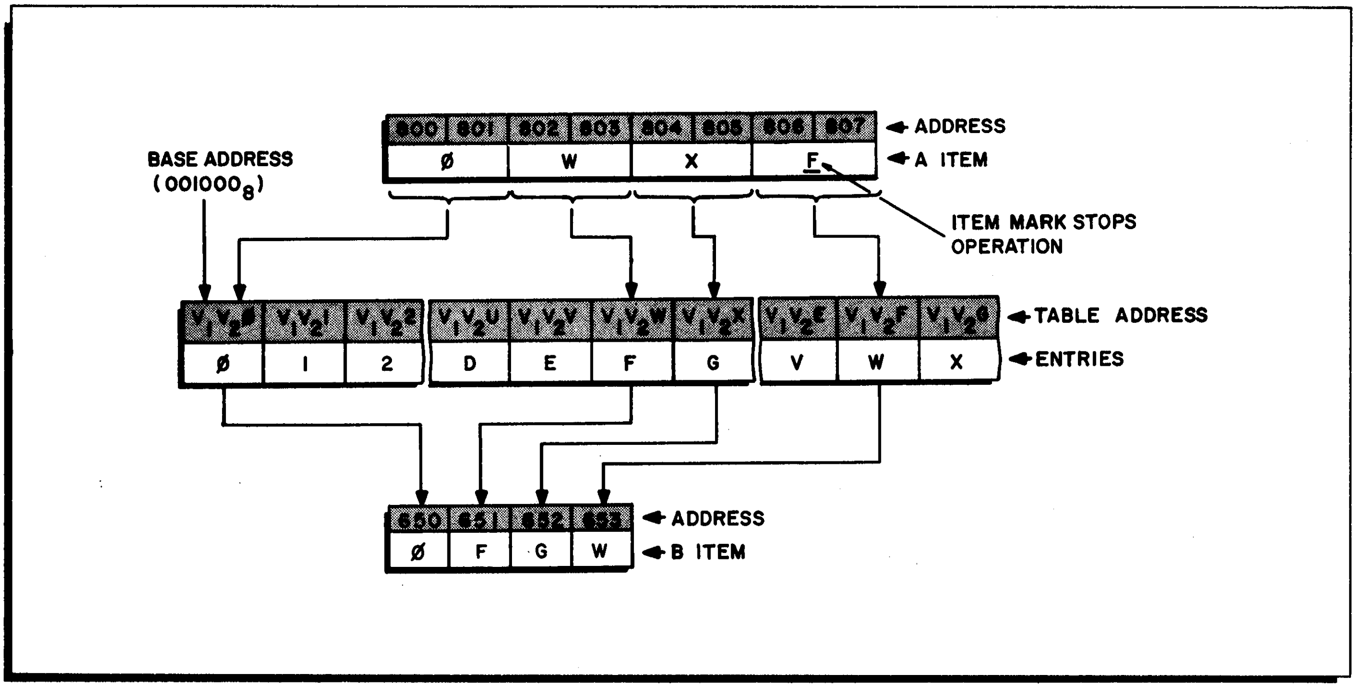
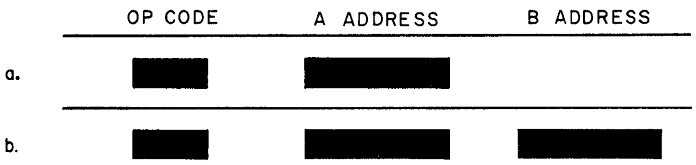


Figure 8-7. MIT Operation

LIB | LOAD INDEX/BARRICADE REGISTER

FORMAT



FEATURES 1114, 1117 AND 1118

FEATURES 1118, 1120 AND 1121

FUNCTION

Format a: Basic storage protection is provided by this instruction format; the character(s) at the location(s) specified by the A address is loaded into the index/barricade register (IBR), specifying the number of 4,096-character main memory banks which are available to a program. The leftmost location of the specified bank is the leftmost location of the protected memory area. (The rightmost location of the protected area is the rightmost location of memory.) For processors other than the Type 4201, the single-character contents of A are loaded into IBR. For the Type 4201 processor, a seventh bit, the rightmost bit of the contents of A-1, is loaded into IBR. The correspondence between the number loaded into IBR and the position of the barricade is in Table 8-18.

Format b: Storage protection with base relocation is provided by this instruction format; the index/barricade register (IBR) is loaded in the same manner as for basic storage protection (format a above), but the barricade is relocated relative to the base relocation address. Consequently, when storage protection is in effect, data cannot be delivered to memory locations above the barricade or below the base relocation address unless processing is in the interrupt mode. The character(s) at the location(s) specified by the B address is loaded into the base relocation register (BRR), specifying the number of 4,096-character main memory banks which are available to a program. The number of main memory locations so designated augments all memory references made in the standard (noninterrupt) mode. For processors other than the Type 4201, the single-character contents of B are loaded into BRR. For the Type 4201 processor, a seventh bit, the rightmost bit of the contents of B-1, is loaded into BRR. The character(s) specified by the A address is loaded into the index/barricade register (IBR), specifying the number of a 4,096-character main memory bank. The barricade is established to the left of the leftmost location in the specified bank (as augmented by the base relocation address). For processors other than the Type 4201, the single-character contents of A are loaded into IBR. For the Type 4201 processor, a seventh bit, the rightmost bit of the contents of A-1, is loaded into IBR. The correspondence between the number loaded into IBR and the position of the barricade is shown in Table 8-18.

Table 8-18. Correspondence Between LIB Setting and Barricade Location

Contents of LIB		Number of Memory Locations to the Left of the Barricade	Contents of LIB		Number of Memory Locations to the Left of the Barricade
Octal	Decimal		Octal	Decimal	
00	0	0	42	34	139,264
01	1	4,096	43	35	143,360
02	2	8,192	44	36	147,456
03	3	12,288	45	37	151,552
04	4	16,384	46	38	155,648
05	5	20,480	47	39	159,744
06	6	24,576	50	40	163,840
07	7	28,672	51	41	167,936
10	8	32,768	52	42	172,032
11	9	36,864	53	43	176,128
12	10	40,960	54	44	180,224
13	11	45,056	55	45	184,320
14	12	49,152	56	46	188,416
15	13	53,248	57	47	192,512
16	14	57,344	60	48	196,608
17	15	61,440	61	49	200,704
20	16	65,536	62	50	204,800
21	17	69,632	63	51	208,896
22	18	73,728	64	52	212,992
23	19	77,824	65	53	217,088
24	20	81,920	66	54	221,184
25	21	86,016	67	55	225,280
26	22	90,112	70	56	229,376
27	23	94,208	71	57	233,472
30	24	98,304	72	58	237,568
31	25	102,400	73	59	241,664
32	26	106,496	74	60	245,760
33	27	110,592	75	61	249,856
34	28	114,688	76	62	253,952
35	29	118,784	77	63	258,048
36	30	122,880	1 00	64	262,144
37	31	126,976	1 01	65	266,240
40	32	131,072	1 02	66	270,336
41	33	135,168	1 03	67	274,432

Table 8-18 (cont). Correspondence Between LIB Setting and Barricade Location

Contents of LIB		Number of Memory Locations to the Left of the Barricade	Contents of LIB		Number of Memory Locations to the Left of the Barricade
Octal	Decimal		Octal	Decimal	
1 04	68	278,528	1 42	98	401,408
1 05	69	282,624	1 43	99	405,504
1 06	70	286,720	1 44	100	409,600
1 07	71	290,816	1 45	101	413,696
1 10	72	294,912	1 46	102	417,792
1 11	73	299,008	1 47	103	421,888
1 12	74	303,104	1 50	104	425,984
1 13	75	307,200	1 51	105	430,080
1 14	76	311,296	1 52	106	434,176
1 15	77	315,392	1 53	107	438,272
1 16	78	319,488	1 54	108	442,368
1 17	79	323,584	1 55	109	446,464
1 20	80	327,680	1 56	110	450,560
1 21	81	331,776	1 57	111	454,656
1 22	82	335,872	1 60	112	458,752
1 23	83	339,968	1 61	113	462,848
1 24	84	344,064	1 62	114	466,944
1 25	85	348,160	1 63	115	471,040
1 26	86	352,256	1 64	116	475,136
1 27	87	356,352	1 65	117	479,232
1 30	88	360,448	1 66	118	483,328
1 31	89	364,544	1 67	119	487,424
1 32	90	368,640	1 70	120	491,520
1 33	91	372,736	1 71	121	495,616
1 34	92	376,832	1 72	122	499,712
1 35	93	380,928	1 73	123	503,808
1 36	94	385,024	1 74	124	507,904
1 37	95	389,120	1 75	125	512,000
1 40	96	393,216	1 76	126	516,096
1 41	97	397,312	1 77	127	520,192

WORD MARKS

Formats a and b:

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	A-2	B_p
Format b:	NXT	A-2	B-2

NOTES

1. The 15 additional index registers which are included in the Storage Protect and Extended Multiprogramming Features are located in the first 60 character locations to the right of the barricade position specified by this instruction. These locations can be used as normal storage locations when they arenot being used for indexing operations.

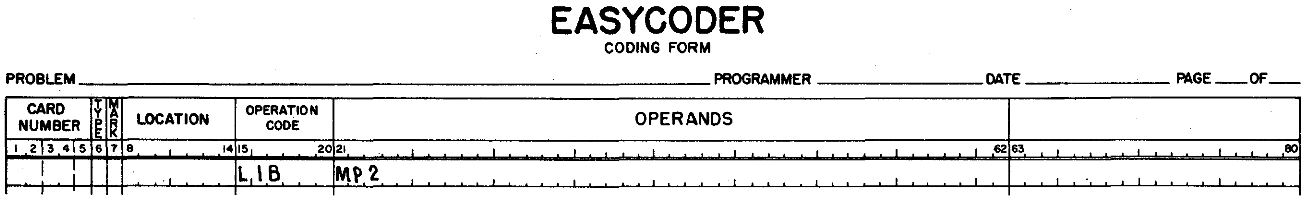
2. The LIB op code is a "privileged" op code which has special significance when storage protection is in effect with the Type 1201, 1251, 2201, or 4201 processor (see Appendix E).

3. This instruction is intended for use in the interrupt mode and should not be issued in the standard mode.

4. The LIB instruction is not interpreted by Easycoder Assembler A, B, or C.

EXAMPLE

Assuming that there are 131,072 storage locations in the main memory, set up the memory in such a way that the "open" memory area consists of locations 0 through 65,535 and the protected memory area consists of locations 65,536 through 131,072. The single octal character "20" is contained in the location tagged MP2.



EASYCODER

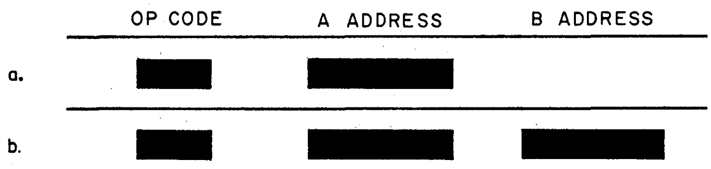
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE __ OF __

CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
1				LIB	MP2	

SIB STORE INDEX/BARRICADE REGISTER

FORMAT



FEATURES 1114, 1117, AND 1118

FEATURES 1118, 1120, AND 1121

FUNCTION

Format a: Basic storage protection is provided by this instruction format; the contents (up to seven bits) of the index/barricade register (IBR) are stored in the character location(s) specified by the A address. All high-order bit positions in A which are not used to specify the contents of the index/barricade register are cleared to zeros. In the Type 4201 processor only, the seventh bit in IBR is stored in the rightmost bit position of location A-1 and the five remaining bit positions in A-1 are cleared to zeros.

Format b: Storage protection with base relocation is provided by this instruction format; the contents of the index/barricade register (IBR) are stored in the same manner as for basic storage protection (format a, above); in addition, the contents of the base relocation register (BRR) are also stored. The contents (up to seven bits) of BRR are stored in the character location(s) specified by the B address. All high-order bit positions in B which are not used to specify the contents of the base relocation register are cleared to zeros. In the Type 4201 processor only, the seventh bit in BRR is stored in the rightmost bit position of location B-1 and the five remaining bit positions in B-1 are cleared to zeros. The contents (up to seven bits) of the index/barricade register are stored in the character location(s) specified by the A address. All high-order bit positions in A which are not used to specify the contents of the index/barricade register are cleared to zeros. In the Type 4201 processor only, the seventh bit in IBR is stored in the rightmost bit position of location A-1 and the five remaining bit positions in A-1 are cleared to zeros.

WORD MARKS

Formats a and b:

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

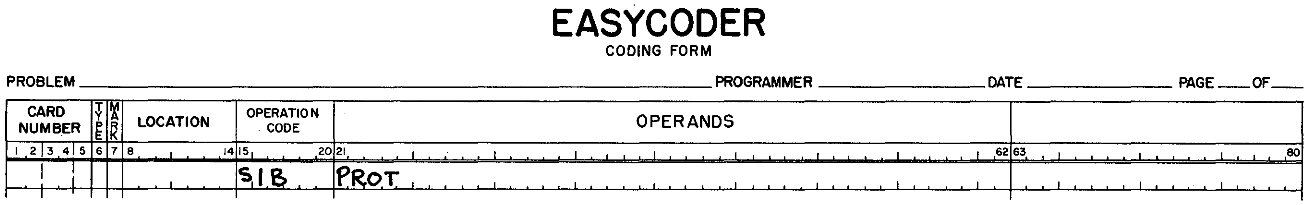
	SR	AAR	BAR
Format a:	NXT	A-2	B_p
Format b:	NXT	A-2	B-2

NOTE

1. The SIB instruction is not interpreted by Easycoder Assembler A, B, or C.
2. This instruction is intended for use in the interrupt mode and should not be issued in the standard mode.

EXAMPLE

Store the contents of the index/barricade register in the single character location tagged PROT.



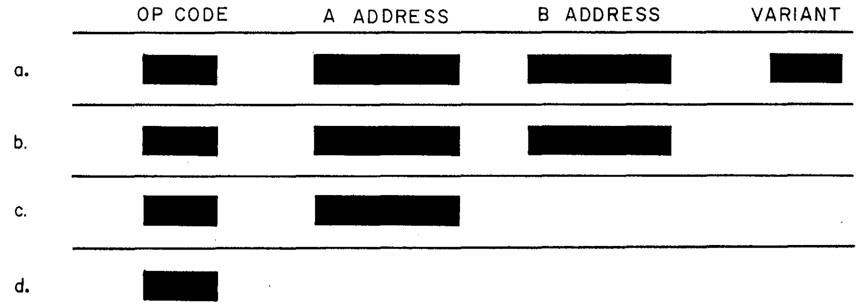
EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
				SIB	PROT	

TLU | TABLE LOOKUP | FEATURE 0191

FORMAT



FUNCTION

Format a: A table in memory is a series of fields, each of which normally contains an argument of a function and the corresponding value of the function (see notes 1 and 2). The Table Lookup instruction initiates a search in a stored table for an argument which bears a specified relationship to a search argument, which is stated in the instruction (see illustration below).

The B address specifies the rightmost location of the stored table, the A address specifies the location of the search argument, and the variant character specifies a relationship (equal to, higher than, etc.) between the desired argument in the table and the search argument. The table is searched from right to left until this relationship is found or until a table field is found which is shorter than the search argument. Then comparison indicators are turned on and the search terminates.

Format b: Search the table whose rightmost location is specified by B for an argument which bears to the search argument specified by A a relationship specified by the variant character of a previous instruction. When this relationship is found or when a table field is found which is shorter than the search argument, turn on comparison indicators and terminate the search.

Format c: Search the table whose rightmost location is specified by the contents of the B-address register (BAR) for an argument which bears to the search argument specified by A a relationship specified by the variant character of a previous instruction. When this relationship is found or when a table field is found which is shorter than the search argument, turn on comparison indicators and terminate the search.

Format d: Search the table whose rightmost location is specified by the contents of BAR for an argument which bears to the search argument specified by the contents of the A-address register (AAR) a relationship specified by the variant character of a previous instruction. When this relationship is found or when a table field is found which is shorter than the search argument, turn on comparison indicators and terminate the search.

WORD MARKS

Formats a, b, c, and d:

The A operand (the search argument) must have a defining word mark. Each table field must also have a defining word mark.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	$A-N_a$	L_{ta}
Format b:	NXT	$A-N_a$	L_{ta}
Format c:	NXT	$A-N_a$	L_{ta}
Format d:	NXT	A_p-N_a	L_{ta}

NOTES

1. Each value in the table is normally stored immediately to the left of the corresponding argument, and each pair (argument plus value) constitutes a field in the table. However, if the values in the table are longer than three characters, it is advisable to store them in another part of memory and to store their 2- or 3-character addresses in the table instead. Since the timing of the TLU instruction depends on the number of characters searched in the table, it is desirable to minimize the length of the table.

2. The Branch on Condition Test instruction (see page 8-35) can be used after Table Lookup to branch to a routine which moves the located value to a work area. Note that at the completion of the TLU instruction, the B-address

register (BAR) contains the address of the desired value (or the address of a location containing the address of the desired value, in the case where the values are too long for efficient storage in the table).

3. The variant characters which specify the desired relationships between the search argument and the argument to be located in the table are as follows:

Variant Character (Octal)	Relationship Which Terminates Search
01	Stored Argument < Search Argument
02	Stored Argument = Search Argument
03	Stored Argument ≤ Search Argument
04	Stored Argument > Search Argument
05	Stored Argument ≠ Search Argument
06	Stored Argument ≥ Search Argument

4. The length of each argument in the table must be equal to the length of the search argument. Note that a short table field (e.g., one which contains a short argument or which contains no value) can be used to terminate the search, which leaves the comparison indicators set to the condition "Stored Argument > Search Argument."

5. The Table Lookup instruction is not interpreted by Easycoder Assembler A, B, or C.

6. Although the Series 200 hardware will chain the variant character of a Table Lookupinstruction, the Mod 2 Assembler permits such chaining only if the B address of the instruction is also chained.

7. Easycoder Assembler D and Mod 2 Assembler:

   a. Format a must use the generic op code (TLU) along with an explicit variant character.

   b. Format b must use a specific op code (e.g., LEH) in order to supply the omitted variant character.

   c. Formats c and d always use the variant character from the previous contents of the variant register. Therefore, the op code used should agree with the one used previously or be the generic form (TLU).

8. The Table Lookup instruction is used by the Series 200 Mod 2 Assembler to implement a number of symbolic statements. The following table indicates the correspondence between the mnemonic op codes for these statements and the TLU variants generated by the Mod 2 Assembler.

Op Code	Table Lookup Statements	Corresponding Variants
LE	Lookup Equal	02
LH	Lookup High	04
LL	Lookup Low	01
LEH	Lookup Equal or High	06
LLE	Lookup Low or Equal	03
LLH	Lookup Low or High (Unequal)	05

EXAMPLE

1. Figure 8-8 shows how a stored table is searched for an argument which bears a specified relationship to a search argument.

Search the table tagged TABLE1 for the value which corresponds to the argument (557) stored in the field tagged ARGMNT.

A Address: ARGMNT (absolute value = location 609)

B Address: TABLE1 (absolute value = location 149)

Variant 1: 02 = (Stored Argument = Search Argument)

EASYCODER
CODING FORM

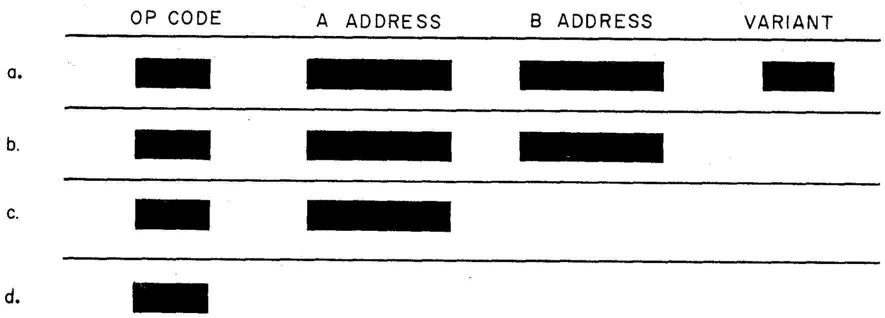
PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				TLU	ARGMNT,TABLE1,02	
2						
3						

BF MOS	MOVE OR SCAN

FEATURE 0191

FORMAT



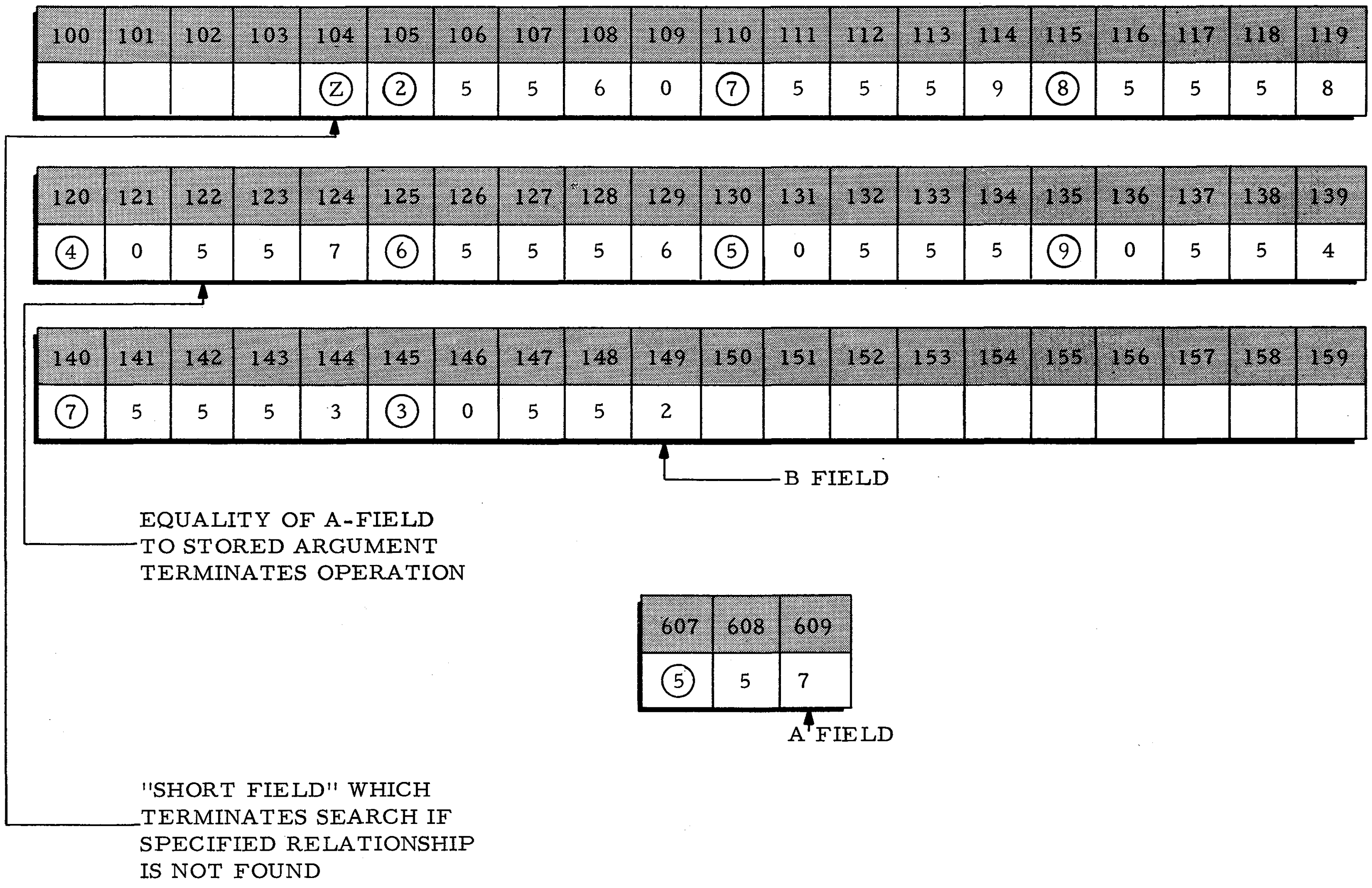
FUNCTION

Format a: The contents of the A field are moved to the B field in the manner specified by the variant character (see Table 8-19). The programmer specifies how the move operation is to be performed by selecting the desired conditions from the table and encoding the resulting two octal digits as the variant character of the instruction. See note 5a.

Format b: This format is valid in symbolic coding only when a specific op code is used to indicate the omitted variant character. The resultant machine-language format and functions are the same as those described for format a.

Format c: The contents of the A field are moved to the field specified by the contents of the B-address register (BAR) in the manner specified by the variant character of a previous instruction (see Table 8-19). See note 5c.

Format d: The contents of the field specified by the contents of the A-address register (AAR) are moved to the field specified by the contents of BAR in the manner specified by the variant character of a previous instruction (see Table 8-19). See note 5c.



100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119
				Ⓩ	②	5	5	6	0	⑦	5	5	5	9	⑧	5	5	5	8

120	121	122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139
④	0	5	5	7	⑥	5	5	5	6	⑤	0	5	5	5	⑨	0	5	5	4

140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
⑦	5	5	5	3	③	0	5	5	2										

607	608	609
⑤	5	7

B FIELD
EQUALITY OF A-FIELD TO STORED ARGUMENT TERMINATES OPERATION
A FIELD
"SHORT FIELD" WHICH TERMINATES SEARCH IF SPECIFIED RELATIONSHIP IS NOT FOUND

Figure 8-8. TLU Operation

Table 8-19. Move or Scan Conditions

Variant Character (Octal)	Condition
X0	No information is moved. The A- and B-address registers are incremented or decremented in accordance with the high-order digit of the variant character.
X1	Move A-field numeric bits to corresponding bit positions in B field.
X2	Move A-field zone bits to corresponding bit positions in B field.
X3	Move A-field data and item-mark bits to corresponding bit positions in B field.
X4	Move A-field word-mark bits to corresponding bit positions in B field.
X5	Move A-field numeric and word-mark bits to corresponding bit positions in B field.
X6	Move A-field zone and word-mark bits to corresponding bit positions in B field.
X7	Move A-field data, word-mark, and item-mark bits to corresponding bit positions in B field.
0X	Move one character from A to B. The A- and B-address registers are decremented by one.
1X	Move characters from left to right (A and B addresses specify leftmost characters in operand fields). Terminate the operation when the first A- or B-field word mark is sensed.
2X	Move characters from right to left (A and B addresses specify the rightmost characters in operand fields). Terminate the operation when the first A-field word mark is sensed.
3X	Move characters from left to right. Terminate the operation when the control character "@" (72_8) is sensed in the A field.
4X	Move characters from right to left. Terminate the operation when the first B-field word mark is sensed.
5X	Move characters from left to right. Terminate the operation when the control character ";" (32_8) with a word mark is sensed in the A field.
6X	Move characters from right to left. Terminate the operation when the first A- or B-field word mark is sensed.
7X	Move characters from left to right. Terminate the operation when either the control character ";" (32_8) with a word mark or control character "@" (72_8) is sensed in the A field.

WORD MARKS

Formats a, b, c, and d:

Word marks and control characters affect the operation of the instruction as described in the table above.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR	
Format a:	NXT	A-1	B-1	VARIANT = 0X
	NXT	$A+N_w$	$B+N_w$	VARIANT = 1X
	NXT	$A-N_a$	$B-N_a$	VARIANT = 2X
	NXT	$A+N_a$	$B+N_a$	VARIANT = (3, 5, or 7)X
	NXT	$A-N_b$	$B-N_b$	VARIANT = 4X
	NXT	$A-N_w$	$B-N_w$	VARIANT = 6X
Format b:	NXT	A-1	B-1	VARIANT = 0X
	NXT	$A+N_w$	$B+N_w$	VARIANT = 1X
	NXT	$A-N_a$	$B-N_a$	VARIANT = 2X
	NXT	$A+N_a$	$B+N_a$	VARIANT = (3, 5, or 7)X
	NXT	$A-N_b$	$B-N_b$	VARIANT = 4X
	NXT	$A-N_w$	$B-N_w$	VARIANT = 6X
Format c:	NXT	A-1	B_p-1	VARIANT = 0X
	NXT	$A+N_w$	B_p+N_w	VARIANT = 1X
	NXT	$A-N_a$	B_p-N_a	VARIANT = 2X
	NXT	$A+N_a$	B_p+N_a	VARIANT = (3, 5, or 7)X
	NXT	$A-N_b$	B_p-N_b	VARIANT = 4X
	NXT	$A-N_w$	B_p-N_w	VARIANT = 6X
Format d:	NXT	A_p-1	B_p-1	VARIANT = 0X
	NXT	A_p+N_w	B_p+N_w	VARIANT = 1X
	NXT	A_p-N_a	B_p-N_a	VARIANT = 2X
	NXT	A_p+N_a	B_p+N_a	VARIANT = (3, 5, or 7)X
	NXT	A_p-N_b	B_p-N_b	VARIANT = 4X
	NXT	A_p-N_w	B_p-N_w	VARIANT = 6X

NOTES:

1. This instruction is available only on the 1201, 1251, 2201, and 4201 (standard) central processors.

2. The character containing the terminating punctuation and/or control characters is moved or scanned in the same manner as the rest of the field.

3. The variant characters and the corresponding mnemonic op codes which they represent are contained in Appendix B.

4. The Move or Scan instruction is not interpreted by the Easycoder Assembler A, B, or C.

5. Although the Series 200 hardware will chain the variant character of a Move or Scan instruction, the Mod 2 Assembler permits such chaining only if the B address of the instruction is also chained.

6. Easycoder Assembler D and Mod 2 Assembler:

   a. Format a must use the generic op code (MOS) with an explicit variant character.

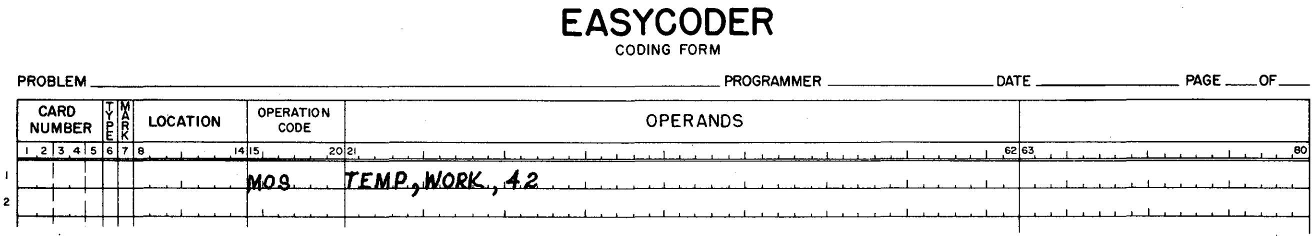
   b. Format b must use a specific op code (MLCW) to supply the omitted variant.

   c. Formats c and d always use the variant character from the previous contents of the variant register. Therefore, the op code used should either agree with the one used previously or be the generic form (MOS).

7. The Move or Scan instruction is used by the Series 200 Mod 2 Assembler to implement a number of symbolic statements. Table B-9 in Appendix B indicates the correspondence between the mnemonic op codes for these statements and the MOS variants generated by the Mod 2 Assembler.

EXAMPLE

Move only the zone bits in the field tagged TEMP to the field tagged WORK from right to left, and terminate the operation when the first word mark in the B field is sensed.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					MOS	TEMP,WORK,42	
2							

INTERRUPT CONTROL

- STORE VARIANT AND INDICATORS
- RESTORE VARIANT AND INDICATORS
- MONITOR CALL
- RESUME NORMAL MODE

SVI | STORE VARIANT AND INDICATORS

FORMAT

OP CODE	A ADDRESS	B ADDRESS	VARIANT
■			■

FUNCTION

The SVI instruction is used to store information regarding the current status of the processor when an interrupt condition occurs. The instruction stores the designated information in up to six consecutive locations following its own variant character.

Each bit in the six-bit variant character ($V_6V_5V_4V_3V_2V_1$) represents processor control registers or indicators whose contents are to be stored in a single character location. The programmer specifies the amount of information to be stored by selecting the desired entries from Table 8-20 and encoding the resulting bit configuration as two octal digits.

Table 8-20. Information Stored by SVI Instruction

VARIANT CHARACTER $V_6V_5V_4V_3V_2V_1$	INFORMATION STORED
X X X X X 1	The contents of the variant register.
X X X X 1 X	The settings of the arithmetic, comparison, address mode, and item-mark trap mode indicators. This information is stored in seven bit positions of the character location — the six data bit positions and the item-mark bit position. The arithmetic and comparison indicators are cleared when their contents have been stored.
X X X 1 X X	The contents of the auxiliary indicators register (AIR). The contents of the arithmetic, comparison, address mode, and item mark trap mode indicators are stored automatically in this register upon the occurrence of an external interrupt. Upon executing an RNM instruction to return to either standard or internal interrupt mode, the specified indicators are reset automatically using the contents of this register. The contents of this register can be changed by using the RVI instruction (see page 8-95). The auxiliary arithmetic and comparison indicators are cleared when their contents have been stored.
X X 1 X X X	The settings of the indicators associated with the scientific unit (see Appendix F) and the sector interrupt masks[2] (see Appendix G). The scientific indicators are cleared when their contents have been stored.

Table 8-20 (cont). Information Stored by SVI Instruction

VARIANT CHARACTER $V_6V_5V_4V_3V_2V_1$	INFORMATION STORED
X 1 X X X X	The settings of the protect,[1] proceed,[1] instruction timeout allow,[2] S-mode, and relocation[2] indicators and (if the processor is in the external interrupt mode) the setting of the internal interrupt (II) mode indicator.[1]
1 X X X X X	The protect, proceed, and instruction time out allow indicators are cleared when their contents are stored. The settings of the interrupt source indicators[1] and the instruction time out indicator. The stored settings of the interrupt source indicators can be tested to determine the status of the processor as follows: 1. Whether the processor is in the external interrupt mode, the internal interrupt mode, or the standard processing mode. 2. The source of the interruption if the processor is in the external interrupt mode; any of three sources can be determined — a peripheral control, the control panel (or console), or the Monitor Call instruction (see page 8-98). 3. Whether an external interrupt (EI) address violation has occurred (if the processor is in the external interrupt mode). 4. Whether an op code violation has occurred (if the processor is in the internal interrupt mode). 5. Whether an internal interrupt (II) address violation has occurred (if the processor is in the internal interrupt mode). The indicators referred to in 3 through 5, above, as well as those which identify the control panel (or console) and the Monitor Call instruction as the interrupt source, are cleared when their contents are stored.

[1] These indicators are included in a Type 1201, 1251, or 2201 processor equipped with the Storage Protect Feature (see Appendix E) or a Type 4201 processor equipped with the Extended Multiprogramming and 8-Bit Transfer Feature (see Appendix G).

[2] These indicators are included in a Type 1201, 1251, 2201, or 4201 processor equipped with the Extended Multiprogramming and 8-Bit Transfer Feature (see Appendix G).

WORD MARKS

A word mark is required in the location following the variant character to terminate the extraction of the SVI instruction. Other word marks (if any) in the locations in which information is stored are ignored and unaffected. Program operation resumes with the next word-marked location following the stored information (the next sequential op code).

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR
NXT	A_p	B_p

VARIANT BIT	STORED CHARACTER LOCATION BITS						
	I/M BIT	B BIT	A BIT	8 BIT	4 BIT	2 BIT	1 BIT
V_1	0	Contents of Variant Register					

V_2	Trap-mode:
	1=yes;
	0=no.
		Address mode:
		01=2-character;
		00=3-character;
		11=4-character.
				Overflow:
				1=yes;
				0=no.
				*
					Zero Balance:
					1=yes;
					0=no. *
						A≤B:
						1=yes;
						0=no.
						*
							A = B:
							1=yes;
							0=no.
							*

V_3	Contents of AIR (identical to information stored by V_2, above)			*	*	*	*
V_4	Extended I/O Indicator I = ON; O = OFF.	MPO: * 1=yes; 0=no.	DVC: * 1=yes; 0=no.	EXO: * 1=yes; 0=no.	Sector 0 Interrupt Mask: 1=on; 0=off.	Sector 1 Interrupt Mask: 1=on; 0=off.	Sector 2 Interrupt Mask: 1 = on; 0 = off.
V_5	0	Protect indicator: 1=on; 0=off. *	Instruction Time-out Allow: 1=on; 0=off.*	S mode	Proceed indicator: 1=on; 0=off. *	Relocation Indicator: 1=on; 0=off.	In external interrupt mode only: 1=II indicator on; otherwise, 0.
V_6	Processor is in external interrupt mode						
	0	EI Address violation: 1=yes; 0=no. *	Monitor Call: 1=yes; 0=no. *	Control panel or console interrupt: 1=yes; 0=no. *	Peripheral interrupt: 1=yes; 0=no.	1	II Mode indicator: 1=on; 0=off.
	Processor is not in external interrupt mode						
	0	II Address violation: 1=yes; 0=no. *	Op code violation: 1=yes; 0=no. *	Instruction Timeout Indicator 1=yes; 0=no. *	0	0	1

*= Indicators that are cleared when their contents are stored.

NOTES

1. Only the number of characters specified by the variant character are stored. They are stored in the order listed in Table 8-20: the contents of the variant register (if specified) are stored in the location immedi-

ately following the SVI instruction, etc., using only those locations actually required to store the requested information.

2. Item-mark and data bit positions which are not used to store information are cleared to zeros.

3. The format in which information is stored by the SVI instruction is shown in the preceding table. Indicators which are cleared (i.e., set to zero) when their contents are stored are indicated by an asterisk (*).

4. Bits corresponding to indicators which are not present in the user's processor are stored as zeros. For instance, an SVI instruction issued in a processor which does not contain the Storage Protect Feature will store zeros in those bit positions which correspond to indicators used only with the Storage Protect Feature.

5. The current status of the arithmetic, comparison, address mode, and trap mode indicators are not stored in the auxiliary indicators register (AIR) when an internal interrupt occurs. The contents of AIR should therefore not be stored by an SVI instruction in the internal interrupt mode, for the contents of AIR would be meaningless at the time of internal interruption.

6. The SVI op code is a "privileged" op code that has special significance when issued in a Type 1201, 1251, 2201, or 4201 processor equipped with the Storage Protect Feature (see Appendix E).

7. This instruction is intended for use in the interrupt mode and should not be issued in the standard mode.

8. This instruction is a standard feature on all processors but the Types 201 and 201-1, on which it is not available.

9. The method of coding interrupt service routines is described in Appendix D, "Interrupt Processing."

10. The contents of the variant register are not altered by the execution of this instruction; i.e., the variant character of the SVI instruction is not stored therein.

EXAMPLE

Store the following information in the three successive memory locations which immediately follow the variant character of the instruction:

1. The contents of the variant register;

2. The contents of the auxiliary indicators register (AIR); and

3. The settings of the interrupt source indicators.

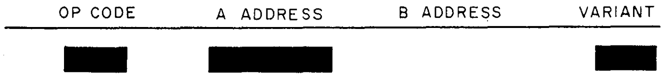
The op code of the SVI instruction is tagged STORE, so that the locations of the stored information are STORE+2, STORE+3, and STORE+4.

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
			STORE	SVI	45	

RVI RESTORE VARIANT AND INDICATORS

FORMAT



FUNCTION

Up to five consecutive characters (previously stored via an SVI instruction) are loaded into the processor control registers and/or indicators specified by the variant character. Characters are retrieved from left to right, beginning with the character specified by the A address.

The low-order five bits of the variant character specify the registers and/or indicators whose contents are to be restored. The programmer specifies the amount of information to be restored by selecting the desired entries from Table 8-21 and encoding the resulting bit configurations as two octal digits.

Table 8-21. Information Restored by RVI Instruction

VARIANT CHARACTER $V_6V_5V_4V_3V_2V_1$	INFORMATION RESTORED
0 X X X X 1	The contents of the variant register.
0 X X X 1 X	The settings of the arithmetic, comparison, address mode, and item-mark trap mode indicators. This information is stored in the six data bits and the item-mark bit of a character location.
0 X X 1 X X	The contents of the auxiliary indicators register (AIR). Upon returning from external interrupt mode to either internal interrupt or standard mode, the contents of this register are moved automatically to the indicators specified above for V_2.
0 X 1 X X X	The settings of the indicators associated with the scientific unit (see Appendix F) and the sector interrupt masks[2] (see Appendix G).
0 1 X X X X	The settings of the protect,[1] proceed,[1] instruction timeout allow,[2] S mode, and relocation[2] indicators and (if the processor is in the external interrupt mode) the setting of the internal interrupt (II) mode indicator.[1]

[1] These indicators are included in a Type 1201, 1251, 2201, or 4201 processor equipped with the Storage Protect Feature (see Appendix E).

[2] These indicators are included in a Type 1201, 1251, 2201, or 4201 processor equipped with Extended Multiprogramming and 8-Bit Transfer Feature (see Appendix G).

WORD MARKS

Word marks neither affect nor are affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR
NXT	A_p	B_p

NOTES

1. Each entry in the righthand column of Table 8-21 is retrieved from a single character location. Only the number of characters corresponding to the selected table entries are retrieved by the RVI instruction.
2. The RVI op code is a "privileged" op code that has special significance when used with a Type 1201, 1251, 2201, or 4201 processor equipped with the Storage Protect Feature (see Appendix E).
3. This instruction is intended for use in the interrupt mode and should not be issued in the standard mode.
4. The format in which information is stored by an SVI instruction is shown in the table on page 8-94. Note that the information contained in the last character location is not restored by the RVI instruction.
5. This instruction is a standard feature on all processors but the Types 201 and 201-1, on which it is not available.
6. The method of coding interrupt service routines is described in Appendix D, "Interrupt Processing."
7. The protect and proceed indicators, when present in the user's system, are not turned on automatically by the computer but instead must be turned on by programmed instructions, as follows: (1) a 1-bit is set in the bit position which, when restored by the RVI instruction, indicates the status of the indicator; and (2) an RVI instruction with a V_5 bit of 1 in the variant character is executed, thereby turning on the appropriate indicator.
8. Unless the contents of the variant register are explictly restored by this instruction, they are not altered by its execution; i.e., the variant character of the RVI instruction is not stored in the variant register.

EXAMPLE

Restore the contents of the variant register and auxiliary indicators register (AIR) that were previously stored by the SVI instruction example on page 8-95.

EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					RVI	STORE+2,Ø5	
2							
3							
4							
5							
6							
7							

MC | MONITOR CALL

FORMAT

OP CODE	A ADDRESS	B ADDRESS
■		

FUNCTION

The Monitor Call instruction causes the processor to enter the external interrupt mode (if the processor is not already in that mode). The following activities are automatically performed:

1. The EI interrupt source indicators are set to show that the Monitor Call instruction is the source of interruption, and the processor enters the external interrupt mode;
2. The settings of the arithmetic, comparison, address mode, and item-mark trap mode indicators are stored in the auxiliary indicators register (AIR);
3. The arithmetic indicators are cleared;
4. The contents of the sequence register (SR) and the external interrupt register (EIR) are interchanged, and the program branches to the instruction whose op code address was previously stored in EIR;
5. The processor switches to the three-character, non-trap mode.

WORD MARKS

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

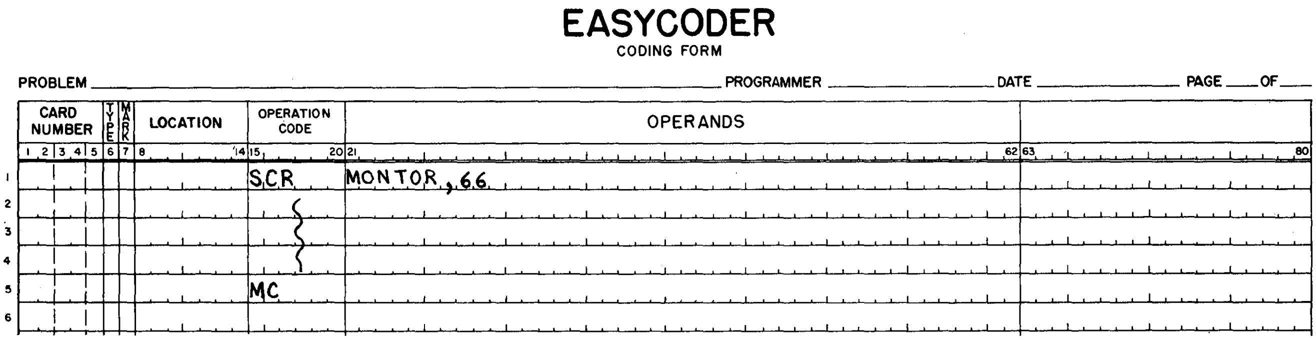
SR	EIR	AAR	BAR
JI (contents of EIR)	NXT	A_p	B_p

NOTES

1. If this instruction is issued in the external interrupt mode, the results are unspecified.
2. The interrupt source indicators can be stored via an SVI instruction (see page 8-92). Their stored contents can then be interrogated by programmed instruction to determine the interrupt source.
3. This instruction is a standard feature on all processors but the Types 201 and 201-1, on which it is not available.

EXAMPLE

Interrupt the central processor and branch to MONTOR, the location of the monitor program. The address tagged MONTOR, was previously stored in EIR.



EASYCODER

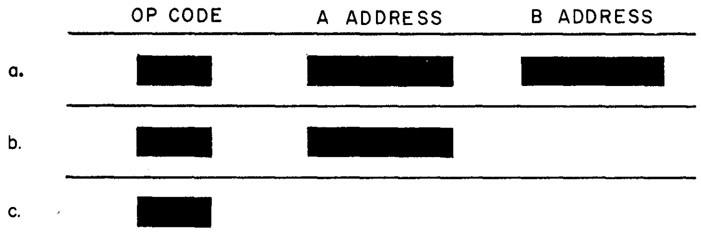
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					SCR	MONTOR,66	
2							
3							
4							
5					MC		
6							

RNM RESUME NORMAL MODE

FORMAT



FUNCTION

Format a: The RNM instruction causes an exit from the program being executed in the interrupt mode (external or internal) to the program which was interrupted. The activities performed depend on the type of interrupt mode in which the instruction is issued.

When the RNM instruction is issued in the external interrupt mode:

1. The EI mode indicators are turned off;
2. The arithmetic, comparison, address mode, and item-mark trap mode indicators are restored to the status specified by the auxiliary indicators register (AIR);
3. The A and B addresses of the RNM instruction are stored in the A- and B-address registers (AAR and BAR), respectively; and
4. The contents of the sequence register (SR) and the external interrupt register (EIR) are interchanged, and the program branches to the instruction whose op code address was initially stored in EIR when the external interrupt occurred.

When the RNM instruction is issued in the internal interrupt mode:

1. The II mode indicator is turned off;
2. The A and B addresses of the RNM instruction are stored in AAR and BAR, respectively; and
3. The contents of SR and the internal interrupt register (IIR) are interchanged, and the program branches to the instruction whose op code address was initially stored in IIR when the internal interrupt occurred.

Format b: This format operates like format a. except that the B address of the RNM instruction is not stored in BAR. The previous contents of BAR are not changed.

Format c: This format operates like format a. except that no instruction addresses are stored. The previous contents of AAR and BAR are not affected by this format.

WORD MARKS

Formats a, b, and c:

Word marks are not affected by this instruction.

ADDRESS REGISTERS AFTER OPERATION

	SR	EIR	IIR	AAR	BAR	
Format a:	NXT	address of op code following RNM instruction	n/a	A	B	RNM ISSUED IN EXTERNAL INTERRUPT MODE
	NXT	n/a	address of op code following RNM instruction	A	B	RNM ISSUED IN INTERNAL INTERRUPT MODE
Format b:	NXT	address of op code following RNM instruction	n/a	A	B_p	RNM ISSUED IN EXTERNAL INTERRUPT MODE
	NXT	n/a	address of op code following RNM instruction	A	B_p	RNM ISSUED IN INTERNAL INTERRUPT MODE
Format c:	NXT	address of op code following RNM instruction	n/a	A_p	B_p	RNM ISSUED IN EXTERNAL INTERRUPT MODE
	NXT	n/a	address of op code following RNM instruction	A_p	B_p	RNM ISSUED IN INTERNAL INTERRUPT MODE

NOTES

1. The address of the instruction which follows the RNM instruction is stored in the appropriate interrupt register (EIR or IIR) when the RNM instruction is executed. This register therefore contains the address of the first instruction executed in the interrupt routine when the next interrupt of the same type occurs. This instruction should be an SVI instruction, which should be the first instruction executed in any interrupt service routine.

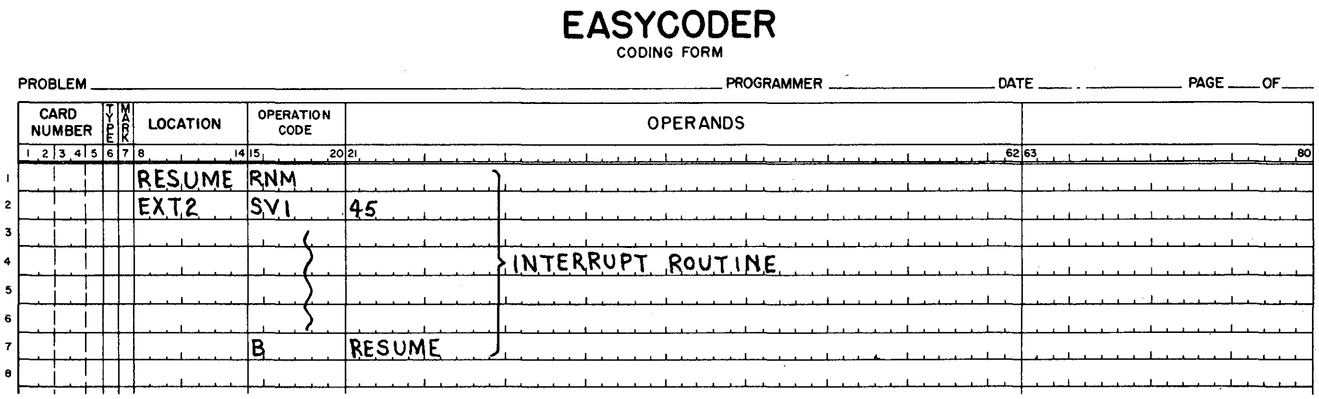
2. The method of coding interrupt service routines is described in Appendix D, "Interrupt Processing."

3. The RNM op code is "privileged" op code which has special significance when used with a Type 1201, 1251, 2201, or 4201 processor equipped with the Storage Protect Feature (see Appendix E).

4. This instruction is intended for use in the interrupt mode and should not be issued in the standard mode.

EXAMPLE

The simplified coding below shows a convenient method of restoring the starting address of the external interrupt routine (EXT2) in EIR when the normal program sequence is resumed.



EASYCODER

CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

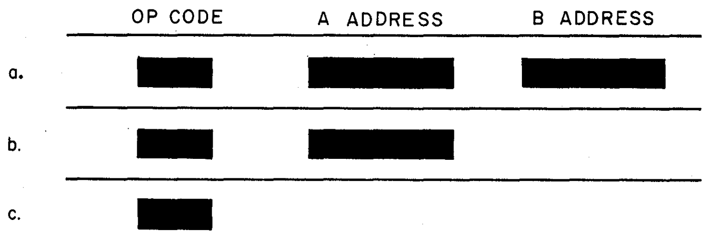
	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1				RESUME	RNM		
2				EXT2	SVI	45	
3							
4						INTERRUPT ROUTINE	
5							
6							
7					B	RESUME	
8							

EDITING

- MOVE CHARACTERS AND EDIT

MCE | MOVE CHARACTERS AND EDIT | FEATURE 013

FORMAT



FUNCTION

Format a: The MCE instruction is used to insert identifying symbols and punctuation and to suppress unwanted zeros in a data field. The A field of an MCE instruction contains the information to be edited. The B field contains an edit control word which provides a framework for the edit operation. When an MCE instruction is executed, the data in the A field is moved to the B field where it is punctuated and formatted according to the edit control word already stored in that field.

NOTE: An LCA instruction can be used to load the control word into the field where the edited information will eventually go. For instance, if the edited information is to be printed, the control word should be loaded into the print image area and the address of this area should be used as the B address of the MCE instruction.

Editing is performed according to the following rules:

RULE 1. Any character in the Series 200 character set can be used in the edit control word. Those characters having special meanings are listed in Table 8-22. Any other character, if included in the edit control word, remains in the edited result in the position where written.

RULE 2. A word mark in the high-order position of the B field controls the edit operation.

RULE 3. The number of replaceable characters in the edit control word must be at least as large as the number of characters in the A field.

RULE 4. Data is transferred from the A field character by character, from right to left. If a zero suppression symbol is not sensed in the edit control word, the edit operation terminates when the B-field word mark is sensed. A zero suppression symbol causes the edited result field to be scanned from left to right. During this scan, high-order zeros and commas are automatically replaced by blanks (unless an asterisk appears immediately to the left of the zero suppression symbol — see rule 5). Zero suppression is terminated by any of the following:

a. a decimal digit from 1 through 9,

b. a decimal point, or

c. the location that initially contained the zero suppression symbol.

RULE 5. An asterisk immediately to the left of the zero suppression symbol in the control word causes high-order zeros and commas to be replaced by asterisks instead of blanks in a zero suppression operation. High-order blanks are also replaced by asterisks.

RULE 6. A dollar sign immediately to the left of the zero suppression symbol in the control word is replaced with an A-field character and causes the edited result to be rescanned following the zero suppression operation. During this scan, the dollar sign is "floated" to the left of the high-order significant digit in the edited result.

Table 8-22. Special Characters in MCE Instruction

CONTROL CHARACTER	FUNCTION
b (blank)	Blanks are replaced with A-field characters such that the rightmost character in the A field replaces the rightmost blank in the edit control word and all higher-order A-field characters replace successively higher-order blanks.
0 (zero)	This symbol specifies zero suppression. Its location in the control word is interpreted as the rightmost limit of zero suppression. It is replaced with an A-field character.
(decimal point)	The decimal point remains in the edited field in the position where written.
, (comma)	Commas remain in the edited field where written unless zero suppression is specified (see rule 4). Commas in control word positions to the left of the high-order character transferred from the A field are replaced by blanks.
C_R, CR (credit) $\bar{0}$ (minus) NOTE: $\bar{0}$ is printed as a minus symbol.	The credit or minus symbol is undisturbed if the sign in the units position of the A field is negative. If the sign is positive, the credit (or minus) symbol is blanked out. A credit (or minus) symbol transferred from the A field is not subject to sign control.
37_8	An octal 37 is replaced by a blank in the edited field.
* (asterisk)	The asterisk remains in the edited field in the position where written unless it appears immediately to the left of the zero suppression symbol (see rule 5).
$ (dollar sign)	The dollar sign remains in the edited field in the position where written unless it appears immediately to the left of the zero suppression symbol (see rule 6).

Format b: The data contents of the A field are edited and stored in the field specified by the contents of the B-address register (BAR) according to the rules outlined above.

Format c: The data field specified by the contents of the A-address register (AAR) are edited and stored in the field specified by the contents of BAR according to the rules outlined above.

WORD MARKS

Formats a, b, and c:

Both the A field and the B field must have defining word marks. The A-field word mark terminates the transfer of data from the A field. The B-field word mark terminates the edit operation if no zero suppression symbol is sensed in the edit control word or if automatic dollar sign insertion is specified in conjunction with zero suppression. The B-field word mark is erased after terminating the edit.

If zero suppression is specified, a word mark is automatically set in the location containing the zero suppression symbol. When this word mark is sensed during the reverse scan associated with the zero suppression operation, it is erased and, if automatic dollar sign insertion is not called for, the edit operation terminates.

ADDRESS REGISTERS AFTER OPERATION

Unspecified

NOTES

1. The zone bits in the units position of the A field are cleared to zero when moved to the B field. Therefore the value of the character in the units position in the A field may change when moved to the B field. For example, an F in the units position of the A field will appear as a 6 in the result field.

2. Floating dollar sign insertion and automatic asterisk insertion can not be performed in the same edit operation.

3. The contents of the variant register are unspecified following the execution of this instruction. Therefore, an instruction requiring a variant character cannot be chained following an MCE instruction.

EXAMPLES[1]

Data Field (A Field)	⓪00009$\bar{9}$
Control Word (B Field)	ⓑbb, bb0. bb&&$\bar{0}$
Result of Edit	.99 ‾

Example 1.

Data Field (A Field)	②5454986
Control Word (B Field)	ⓑbb&bb&bbb
Result of Edit	254 54 986

Example 2.

Data Field (A Field)	⓪0045$\overset{+}{0}$
Control Word (B Field)	$\textcircled{\$}$b, bb0. bb&CR*
Result of Edit	$ 4.50 *

Example 3.

[1] The character (37_8) is shown as an ampersand (&) in these examples. However, the ampersand is not the only equivalent of 37_8 as shown in Table B-6.

Data Field (A Field)	⓪0897445
Control Word (B Field)	ⓑbbb, b$0. bb
Result of Edit	$8, 974. 45

Example 4.

Data Field (A Field)	⓪010450
Control Word (B Field)	ⓑb, b*0. bb
Result of Edit	***104. 50

Example 5.

INPUT/OUTPUT

- PERIPHERAL DATA TRANSFER
- PERIPHERAL CONTROL AND BRANCH

INPUT/OUTPUT CONTROL OPERATIONS

Effective control over data transfers between the central processor and peripheral units and over the peripheral units themselves is maintained by the use of two basic instructions: Peripheral Data Transfer (PDT), and Peripheral Control and Branch (PCB). The PDT instruction is used to initiate data transfer operations and certain other related operations, such as backspace magnetic tape and advance the printer form.

The PCB instruction can perform four distinct functions: (1) it initiates strictly mechanical (non-data transfer) operations such as magnetic tape rewinds and card rejections; (2) it causes a program branch to be performed contingent upon the settings of peripheral condition indicators; (3) it changes the operational mode of a peripheral control; and (4) it allows a peripheral control to interrupt (or directs the control not to interrupt) the central processor when data transfer is completed.

Detailed programming and operating information for Series 200 peripheral devices is provided in separate publications. The remainder of this section is a summary of the PDT and PCB instructions, based on the assumption that the user is familiar with the contents of the applicable documents. In all applicable cases, the coding summary for a device is followed by a reference to the specific Honeywell manual or information bulletin where additional information can be found.

SELECTING RWC ASSIGNMENTS FOR USE IN PDT INSTRUCTIONS

As described below, the first control character (C1) in a PDT instruction is referred to as the "read/write channel assignment." This six-bit character specifies the read/write channel(s) selected to complete the data path (see also pages 1-16 and 2-13). When coding a PDT instruction, the programmer may enter Table 8-24 to select an RWC assignment. The following discussion concerns the considerations involved in selecting RWC assignments and the correspondence between achievable data transfer rates and RWC assignments.

Considerations in Selecting RWC Assignments

At least four factors must be considered when selecting an RWC assignment. These factors are: (1) the data transfer rate of the device being addressed; (2) the processor being used; (3) the I/O sector to which the device is attached; and (4) the necessity of being upward compatible.

DEVICE DATA TRANSFER RATE

The first consideration in selecting an RWC assignment is the rated speed at which the device being addressed transfers data to or from main memory. The one or more RWC's assigned to an operation must receive memory accesses often enough to keep up with the I/O data transfer rate of the device. For example, the RWC assignment used in a PDT instruction which addresses a Type 258 Disk Pack Drive must designate a data transfer capacity high enough to keep pace with the device's 208, 000-character-per-second transfer rate.

However, due to mechanical tolerances, some devices may transfer data at instantaneous rates higher than their nominal transfer rates. In a few such cases, the devices require an RWC assignment having a greater data handling capacity than would be required if the nominal data transfer rate were maintained. As an example, a Type 204B-5 tape drive using a density of 556 bits per inch requires an RWC assignment having a data handling capacity of 167, 000 characters per second, even though the nominal transfer rate for this device is less than 83, 300 characters per second.

Table 8-23 lists the minimum RWC capacity requirements for each Series 200 peripheral device.

Table 8-23. Minimum RWC Capacity Requirements for Series 200 Peripheral Devices

DEVICE	MINIMUM RWC CAPACITY REQUIRED (characters per second)
204A-1 Magnetic Tape Unit	83.3 KC
204A-2 Magnetic Tape Unit	167 KC
204A-3 Magnetic Tape Unit	167 KC
204B-1, -2 Magnetic Tape Units	
200/556bpi	83.3 KC
204B-3, -4 Magnetic Tape Units	
200/556 bpi	83.3 KC
204B-5 Magnetic Tape Unit	
200 bpi	83.3 KC
556 bpi	167 KC
204B-7 Magnetic Tape Unit	
200/556/800 bpi	83.3 KC
204B-8 Magnetic Tape Unit	
200/556 bpi	83.3 KC
800 bpi	167 KC
204B-9 Magnetic Tape Unit	
200/556 bpi	83.3 KC
800 bpi	167 KC
1200 bpi	167 KC
204B-11, -12 Magnetic Tape Units	
200/556 bpi	83.3 KC
204C-13, -14 Magnetic Tape Units	83.3 KC
206 Printer	167 KC
214-1 Card Punch	83.3 KC

Table 8-23 (cont). Minimum RWC Capacity Requirements for Series 200 Peripheral Devices

DEVICE	MINIMUM RWC CAPACITY REQUIRED (characters per second)
214-2 Card Reader/Punch	
Read	83.3 KC
Punch	83.3 KC
222 Printers (All Models)[1]	167 KC
223 Card Reader	83.3 KC
223-2 Card Reader	83.3 KC
224-1, -2 Card Reader/Punch	
Read	83.3 KC
Punch	83.3 KC
227 Card Reader-Card Punch	
Read	167 KC
Punch	167 KC
232 MICR Reader-Sorter and Control	83.3 KC
233-2 MICR Control	83.3 KC
209 Paper Tape Reader	83.3 KC
209-2 Paper Tape Reader	83.3 KC
210 Paper Tape Punch	83.3 KC
212 On-Line Adapter	167 KC
212-1 Central Processor Adapter	167 KC
213-4 Time-of-Day Clock	83.3 KC
220-1, -2, -3 Consoles	83.3 KC
234 Calcomp Plotter Control	83.3 KC
235 Optical Journal Reader Control	83.3 KC
237 Bill Feed Printer Control	167 KC
258 Disk Pack Drive	250 KC
259 Disk Pack Drive	250 KC
259A Disk Pack Drive	167 KC
259B Disk Pack Drive	167 KC
261 Disk File	250 KC
262 Disk File	250 KC
270 Random Access Drum Storage (All Models)	167 KC
281 Single-Channel Communication Controls[2] (All Models)	83.3 KC
286-1, -2, -3 Multi-Channel Communication Controls	83.3 KC
286-4, -5 Message-Mode, Multi-Channel Communication Controls[3]	83.3 KC
287 AUTODIN Communication Control[2]	83.3 KC
287-1 USASCII AUTODIN Communication Control[2]	83.3 KC

[1] When a 222-3, -4, -5, or -6 printer is equipped with the Print Buffer (Feature 036), the transfer rate must be either 83.3 KC or 167 KC.

[2] The 281-2F, 287, and 287-1 controls require exclusive assignment of two 83.3 KC RWC's when operating in full-duplex mode.

[3] The maximum RWC capacity that can be assigned to a 286-4 or 286-5 is 167 KC.

THE PROCESSOR BEING USED

Each Series 200 processor except the 1201 and 1251 comes with a basic and an expanded I/O configuration. These I/O configurations include different numbers of RWC's. Clearly, then, the identity of the processor being used and whether or not it is an expanded configuration will help to determine what RWC assignments are available for use. For example, in the basic (3-channel) Type 201-2 processor, eight RWC assignments are available. Input/output operations proceeding at rates up to 167,000 characters per second can be handled on individual channels by designating either of two RWC assignments available for each channel. Two RWC assignments are provided for interlocking channels to handle rates of up to 333,000 and 500,000 characters per second, respectively. Adding Feature 016 to a Type 201-2 allows the use of two additional RWC assignments: one to increase I/O flexibility by permitting a fourth simultaneous I/O operation, and the other to interlock two channels in such a way as to achieve a 250,000-character-per-second transfer rate. Note that the maximum data transfer rate (all channels) achievable with the expanded I/O configuration remains 500,000 characters per second.

As indicated in Section II, Type 4201 processors are equipped with variable-speed read/write channels. No more than two RWC's (a primary and the corresponding auxiliary) are ever made busy by a single RWC assignment. RWC's not made busy by a high-speed transfer are available for use in other operations. For example, in a basic 4201, a 250,000-character-per-second transfer from an I/O device in sector 1 can be handled using RWC assignment 55_8 and only RWC 3 will be tied up. The other three sector 1 RWC's will still be available for use in other operations, e.g., three 83,300-character-per-second transfers.

The "sector escape" code feature of the Type 4201 (used in both PDT and PCB instructions) makes variable-speed read/write channels even more attractive. An escape code allows an RWC normally restricted to operating in one sector to be used for I/O transfers in another sector. For example, an escape code can be used to assign RWC 1, normally used only in sector 1, to a data transfer in sector 2.

This facility enables the programmer to transfer RWC's temporarily to a sector performing several low-speed operations from another sector in which one or two operations are using the sector's entire data handling capacity. For example, escape codes could be used in a basic 4201 to perform simultaneously the operations indicated in Figure 8-9. In this example, escape codes are used to enable RWC's 1 and 1' to operate in sector 2.

Operation	Sector	Data Transfer Rate of Operation	RWC Used
1	1	167,000 char/sec	2
2	1	333,000 char/sec	3
3	2	83,000 char/sec	1
4	2	83,000 char/sec	1'
5	2	83,000 char/sec	4
6	2	83,000 char/sec	4'
7	2	83,000 char/sec	5
8	2	83,000 char/sec	6

Figure 8-9. Example of Operation Utilizing Escape Codes

INPUT/OUTPUT SECTOR TO WHICH DEVICE IS CONNECTED

Each input/output sector in a Series 200 processor has a maximum total data transfer capacity. For Model 200, 1200, 1250, and 2200 processors, this maximum is 500,000 characters per second. Sector 3 of an expanded 4201 processor can handle up to 333,000 characters per second.

The identity of the I/O sector to which the addressed device is connected also becomes a factor when selecting RWC assignments for expanded Type 2201 processors and for all 1251 and 4201 processors. In general, the RWC assigned to an operation should be associated with the sector to which the addressed device is connected. However, as indicated above, this rule can be circumvented to advantage in 4201 processors by the use of escape codes.

UPWARD COMPATIBILITY

Because of the manner in which upward compatibility has been consistently implemented in Series 200 processors, very little consideration need be given to this factor when selecting RWC assignment codes. The one case where such consideration must be given is when assigning a primary RWC for which there is no corresponding auxiliary channel in the processor being programmed to an operation faster than 83,300 characters per second. An example of such a case is the assignment of the single channel RWC 1 to a drum read operation (102,000 characters per second) to be performed in a basic (3-channel) Model 200 processor. In the basic processor, RWC 1 can handle transfer rates up to 167,000 characters per second. However, in an expanded Model 200, RWC's 1 and 1' can handle only 83,300 characters per second apiece unless they are interlocked. Thus, if the attempt were made to run the basic 200 program on an expanded 200, the RWC 1 alone would not be able to handle the drum's transfer rate.

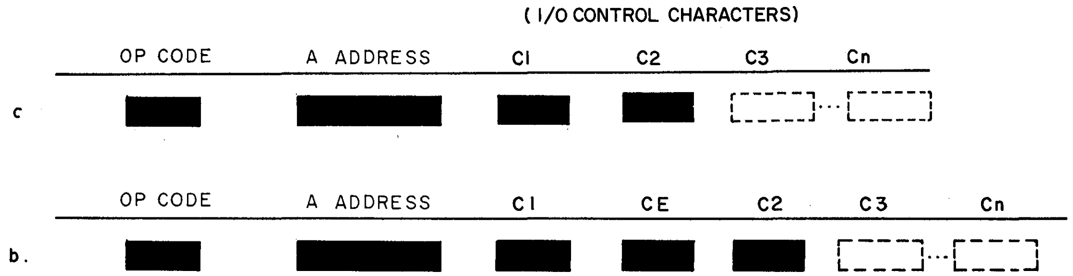
In order to avoid such problems, the following general rule should be followed:

> The RWC assignment in a PDT instruction addressing a device which operates between 83,300 and 167,000 characters per second should be such that it would interlock the primary channel and its auxiliary if the program were run in a processor equipped with both channels; i.e., its high-order digit should be 5 or 7.

Clearly, there is no need to specify the "interlock" assignment if the device runs slower than 83,300 characters per second. Rather, in the interest of making more RWC's available for use in other operations, it is often wise in such cases to specify the single-channel assignment.

PDT | PERIPHERAL DATA TRANSFER

FORMAT



FUNCTION

Format a: The PDT instruction causes data to be transferred between a peripheral device and the main memory area whose leftmost location is designated by the A address. Data transfer is terminated according to the data medium employed. Input/output control characters specify the data path through which the transfer is to be accomplished, as indicated in Tables 8-24 and 8-26.

Format b: Data is transferred between a peripheral device and the main memory area whose leftmost location is designated by the A address. Data transfer is terminated according to the data medium employed. Input/output control characters and an escape code specify the data path through which the transfer is to be accomplished, as indicated in Tables 8-24, 8-25, and 8-26.

Table 8-24. Description of PDT I/O Control Character C1 (Read/Write Channel Assignment)

Configuration	Data Handling Capacity (Char./Sec.)	RWC's Interlocked and Made Busy[1]	RWC Assignment Code
	Type 201, 201-1 Processors		
Basic	167,000	$\underline{1}$	51
	167,000	$\underline{2}$	52
	167,000	$\underline{3}$	53
With Feature 016	83,000	$\underline{1}$	11
	83,000	$\underline{1'}$	15
	167,000	$\underline{1}$, 1'	51
	Type 201-2 Processor		
Basic	167,000	$\underline{1}$	51
	167,000	$\underline{2}$	52
	167,000	$\underline{3}$	53
	333,000	2, $\underline{3}$	56
	500,000	1, 2, $\underline{3}$[2]	54
With Feature 016	83,300	$\underline{1}$	11
	83,300	$\underline{1'}$	15
	167,000	$\underline{1}$, 1'	51
	250,000	1', $\underline{3}$	55
	Type 1201 Processor		
	Same as Type 201-2 with Feature 016.		
	Type 1251 Processor		
	Sector 1		
Basic	83,300	$\underline{1}$	11
	83,300	$\underline{1'}$	15
	167,000	$\underline{1}$, 1'	51
	167,000	$\underline{2}$	52
	167,000	$\underline{3}$	53
	250,000	1', $\underline{3}$[3]	55
	333,000	2, $\underline{3}$[3]	56
	500,000	1, 1', 2, $\underline{3}$[3]	54
	Sector 2		
With Feature 1115	83,300	$\underline{4}$	31
	83,300	$\underline{4'}$	35
	167,000	$\underline{4}$, 4'	71
	167,000	$\underline{5}$	72
	167,000	$\underline{6}$	73
	250,000	4', $\underline{6}$[4]	75
	333,000	5, $\underline{6}$[4]	76
	500,000	4, 4', 5, $\underline{6}$[4]	74
	Type 2201 Processor		
Basic	Same as Type 201-2 with Feature 016.		
With Feature 1115	Sector 1 same as Type 201-2 with Feature 016. Sector 2 same as Type 1251.		

Table 8-24 (cont). Description of PDT I/O Control Character C1 (Read/Write Channel Assignment)

Configuration	Data Handling Capacity (Char./Sec.)	RWC's Interlocked and Made Busy[1]	RWC Assignment Code
	Type 4201 Processor		
Basic	Sector 1 same as Type 201-2 with Feature 016.		
	Sector 2 same as Type 2201 with Feature 1115.		
With Feature 1116	Sector 1		
	83,300	1	11
	83,000	2	12
	83,000	3	13
	83,000	1'	15
	83,000	2'	16
	83,000	3'	17
	250,000	2	50
	167,000	1, 1'	51
	167,000	2, 2'	52
	167,000	3, 3'	53
	250,000	3	55
	333,000	3	56
	500,000	3	54
	Sector 2		
	83,300	4	31
	83,000	5	32
	83,000	6	33
	83,000	4'	35
	83,000	5'	36
	83,000	6'	37
	250,000	5	70
	167,000	4, 4'	71
	167,000	5, 5'	72
	167,000	6, 6'	73
	250,000	6	75
	333,000	6	76
	500,000	6	74
	Sector 3		
	83,300	8	22
	83,000	9	23
	83,000	8'	26
	83,000	9'	27
	167,000	8, 8'	62
	167,000	9, 9'	63
	333,000	9	66

[1] Underlined numbers identify the RWC whose corresponding starting and current location counters (SLC and CLC) are used in the operation.

[2] In processors equipped with RWC 1', that channel is also interlocked.

[3] Uses RWC 3 for address storage during data transfer.

[4] Uses RWC 6 for address storage during data transfer.

Note: RWC 2 cannot be active while RWC 5 is active, nor can RWC 3 be active while RWC 6 is active.

Escape Code (CE)

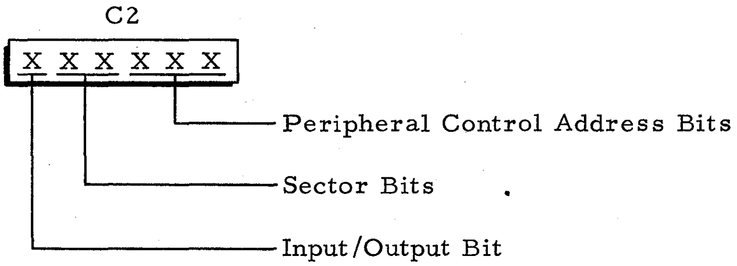
The escape code is part of format b. of the PDT instruction. If the second control character is one of the escape codes shown in Table 8-25, the read/write channel(s) designated by C1 is assigned to an I/O operation in the sector indicated by the escape code. The addressed device must be connected to this sector.

Table 8-25. Description of PDT I/O Character CE (Escape Code)

Escape Code	Sector to Which RWC is Assigned
10	Sector 1
12	Sector 2
13	Sector 3

Table 8-26. Description of PDT I/O Control Character C2 (Peripheral Control Designation)

<table>
<tr><th>CONTROL CHARACTER</th><th>DESCRIPTION</th></tr>
<tr><td>C2</td><td>
PERIPHERAL CONTROL DESIGNATION: This six-bit character specifies the logical address of the peripheral control to be used in the data transfer.



Input/Output Bit: This bit specifies the direction of data transfer when a peripheral control capable of both reading and writing is involved in the transfer. When such a bidirectional control is used,

0 = transfer data from memory to the peripheral control (output),
1 = transfer data to memory from the peripheral control (input).

Specific communication controls and the Type 212 On-Line Adapter are exceptions to this rule (see Table 8-27).

The input/output bit can be either zero or one in the logical address of unidirectional peripheral control (e. g., a printer). However, if compatibility with the Type 8201 processor is desired, 1 and 0 must be specified, respectively, for input and output devices.

Sector Bits: These bits apply only to the Models 1250, 2200, and 4200 and specify the sector in which the peripheral control is connected. They are specified as follows:

<table>
<tr><th></th><th>Models 1250 and 2200</th><th>Model 4200</th></tr>
<tr><td>Sector 1</td><td>0 0</td><td>0 0</td></tr>
<tr><td>Sector 2</td><td>1 0</td><td>1 0</td></tr>
<tr><td>Sector 3</td><td>---</td><td>1 1</td></tr>
</table>
</td></tr>
</table>

Table 8-26 (cont). Description of PDT I/O Control Character C2 (Peripheral Control Designation)

<table>
<tr><th>CONTROL CHARACTER</th><th>DESCRIPTION</th></tr>
<tr><td>C2
(cont)</td><td>Sector bits must always be zeros in Model 200 and 1200 peripheral addresses.

Peripheral Control Address Bits: These three bits, in conjunction with the preceding three bits, identify the address of the peripheral control involved in the operation. It is recommended that the following octal configurations be used for control character C2 in order to provide uniformity among Series 200 installations:
<table>
<tr><th>Peripheral Control</th><th>Octal Address[1]</th></tr>
<tr><td>Magnetic Tape Control[2]</td><td>00 (output)
40 (input)</td></tr>
<tr><td>Paper Tape Reader or Card Reader[3]</td><td>41</td></tr>
<tr><td>Paper Tape Punch or Card Punch[3]</td><td>01</td></tr>
<tr><td>Printer</td><td>02</td></tr>
<tr><td>Type 212 On-Line Adapter</td><td>42</td></tr>
<tr><td>Console</td><td>07 (output)
47 (input)</td></tr>
<tr><td>Disk Control</td><td>04 (output)
44 (input)</td></tr>
</table>
</td></tr>
<tr><td colspan="2">[1] C2 configurations are made up of (1) the input/output bit and (2) the peripheral control address bits. In Series 200 systems in which sector designations apply (viz., the Models 1250, 2200, and 4200), the specification of the sector bits may alter these recommended configurations.

[2] In Series 200 installations containing both 1/2-inch and 3/4-inch magnetic tape systems, the recommended addresses of 00 and 40 should be assigned to the 1/2-inch tape control.

[3] In Series 200 installations containing a card reader/punch unit, these recommended addresses apply. However, if the installation contains a second card reader, the reader portion of the card reader/punch should be assigned the address 43_8 and the second card reader assigned the address 41_8.</td></tr>
</table>

Additional Parameters (C3 through Cn)

The specific use of these control characters is dependent upon the type of peripheral device addressed. A summary of coding for these characters may be found in Tables 8-27 through 8-33.

PUNCTUATION MARKS

The execution of this instruction neither affects nor is affected by word marks or item marks. However, record marks may terminate the data transfer, depending upon the device used and the operation performed (see the specific Honeywell publications).

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR
NXT	A	B_p

NOTES

1. If either the read/write channel or the peripheral control (specified by C1 and C2, respectively) is found "busy" during the extraction of a PDT instruction, the instruction is re-extracted: the contents of SR are set back to the address of the PDT op code, and the extraction process begins again. This process, which allows the processor to respond to interrupt signals that may occur while the PDT instruction is awaiting the availability of a read/write channel or peripheral control, is not performed in the Type 201 and 201-1 processors; PDT extraction in these two processors waits until the busy channel or control is available.

2. The PDT op code is a "privileged" op code when used in a Type 1201, 1251, 2201, or 4201 processor equipped with the Storage Protect Feature (see Appendix E).

3. Format b. of the PDT instruction is applicable only to Type 4201 processors.

4. Unspecified central processor activity can occur when an attempt is made to execute a PDT instruction having a read/write channel assignment (C1) of zero. It is therefore imperative that every PDT instruction contain some legal RWC assignment.

5. Control character C1 of a PDT instruction is stored in the variant register.

EXAMPLE

Read a card into the 80-character image area tagged CREAD. Use RWC2 and assume that the card reader control is assigned to the logical address of octal 41. Note that the data transfer rate in a card reading operation is less than 83,300 characters per second.

EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					PDT	CREAD,12,41	
2							

Table 8-27. Summary of PDT I/O Control Characters

INPUT/OUTPUT OPERATION		PDT I/O CONTROL CHARACTER					
		C1 READ/WRITE CHANNEL	C2 CONTROL UNIT	C3 ADDITIONAL PARAMETERS	C4 ADDITIONAL PARAMETERS	C5 ADDITIONAL PARAMETERS	C6 ADDITIONAL PARAMETERS
CARD	READ	X X	X X	none	none	none	none
	PUNCH	X X	X X	none	none	none	none
See: Type 223 Card Reader (Order No. 504), Type 214-1 Card Punch (Order No. 451), Type 214-2 Card Reader/Punch (Order No. 432), Type 224 Card Reader/Punch (Order No. 506) or Type 227 Card Reader/Punch (Order No. 564)							
PAPER TAPE	READ	X X	X X	See Table 8-28 (page 8-124)	none	none	none
	PUNCH	X X	X X	See Table 8-29 (page 8-124)	none	none	none
See: Types 209, 209-2, and 210 Paper Tape Equipment (Order No. 507)							
PRINTER	PRINT	X X	X X	See Table 8-30 (page 8-125)	none	none	none
See: Type 222 Printers (Order No. 562)							

Table 8-27 (cont). Summary of PDT I/O Control Characters

INPUT/OUTPUT OPERATION		C1 READ/WRITE CHANNEL	C2 CONTROL UNIT	C3 ADDITIONAL PARAMETERS	C4 ADDITIONAL PARAMETERS	C5 ADDITIONAL PARAMETERS	C6 ADDITIONAL PARAMETERS
MAGNETIC TAPE 1/2-INCH	READ FORWARD	X X	X^1 X	6 D^3 (D=tape drive, 0 - $7)^6$	none	none	none
	READ REVERSE (Feature 010 or 011)	X X	X^1 X	2 D^4 (D=tape drive, 0 - $7)^6$	none	none	none
	WRITE	X X	X^2 X	2 D^5 (D=tape drive, 0 - $7)^6$	none	none	none
	SPACE FORWARD	X X	X^1 X	4 D (D=tape drive, 0 - $7)^6$	none	none	none
	BACKSPACE	X X	X^1 X	0 D (D=tape drive, 0 - $7)^6$	none	none	none
	ERASE	X X	X^2 X	0 D (D=tape drive, 0 - $7)^6$	none	none	none
See: Type 204B Series Magnetic Tape Unit (Order No. 503), Types 204B-11 and 204B-12 Magnetic Tape Units (Order No. 502), or Types 204C-13 and 204C-14 Magnetic Tape Units (Order No. 623)							
MAGNETIC TAPE 3/4-INCH	READ FORWARD	X X	X^1 X	6 D (D=tape drive, 0 - 3)	none	none	none
	READ SUPPRESSING CHANNEL	X X	X^1 X	5 D (D=tape drive, 0 - 3)	C 0 (C=channel to be suppressed)	none	none
	WRITE	X X	X^2 X	6 D (D=tape drive, 0 - 3)	none	none	none
	SKIP WRITE	X X	X^2 X	4 D (D=tape drive, 0 - 3)	none	none	none
	BACKSPACE	X X	X^1 X	0 D (D=tape drive, 0 - 3)	none	none	none
See: Honeywell Series 200 Equipment Operators' Manual (Order No. 040), or Type 204A Series Magnetic Tape Units (Order No. 863)							
RANDOM ACCESS DRUM	SEARCH AND READ	X X	X^1 X	See Table 8-29 (page 8-117)	$\underbrace{\text{0 T} \qquad\qquad \text{T T}}$ 9-bit track address numbered 0 - 777 (octal)		SS Sector address numbered 0 - 47 (octal)
	READ	X X	X^1 X	See Table 8-31 (page 8-125)	none	none	none
	SEARCH AND WRITE	X X	X^2 X	See Table 8-31 (page 8-125)	$\underbrace{\text{0 T} \qquad\qquad \text{T T}}$ 9-bit track address numbered 0 - 777 (octal)		S S Sector address numbered 0 - 47 (octal)
	WRITE	X X	X^2 X	See Table 8-31 (page 8-125)	none	none	none
	READ ADDRESS REGISTER	X X	X^1 X	See Table 8-31 (page 8-125)	none	none	none
See: Type 270 Random Access Drum and Control (Order No. 009)							

Table 8-27 (cont). Summary of PDT I/O Control Characters

INPUT/OUTPUT OPERATION		PDT I/O CONTROL CHARACTER					
		C1 READ/WRITE CHANNEL	C2 CONTROL UNIT	C3 ADDITIONAL PARAMETERS	C4 ADDITIONAL PARAMETERS	C5 ADDITIONAL PARAMETERS	C6 ADDITIONAL PARAMETERS
DISK DEVICES	LOAD ADDRESS REGISTER	X X	X^2 X	0 4	none	none	none
	STORE ADDRESS REGISTER	X X	X^1 X	0 4	none	none	none
	WRITE INITIAL	X X	X^2 X	0 0 or 1 0 *	none	none	none
	EXTENDED WRITE INITIAL	X X	X^2 X	2 0 or 3 0 *	none	none	none
	WRITE	X X	X^2 X	0 1 or 1 1 *	none	none	none
	EXTENDED WRITE	X X	X^2 X	2 1 or 3 1 *	none	none	none
	SEARCH AND WRITE	X X	X^2 X	0 2 or 1 2 *	none	none	none
	EXTENDED SEARCH AND WRITE	X X	X^2 X	2 2 or 3 2 *	none	none	none
	SEARCH AND WRITE NEXT	X X	X^2 X	0 3 or 1 3 *	none	none	none
	EXTENDED SEARCH AND WRITE NEXT	X X	X^2 X	2 3 or 3 3 *	none	none	none
	SEARCH AND READ	X X	X^1 X	0 2 or 1 2 *	none	none	none
	EXTENDED SEARCH AND READ	X X	X^1 X	2 2 or 3 2 *	none	none	none
	SEARCH AND READ NEXT	X X	X^1 X	0 3 or 1 3 *	none	none	none
	EXTENDED SEARCH AND READ NEXT	X X	X^1 X	2 3 or 3 3 *	none	none	none
	READ INITIAL	X X	X^1 X	0 0 or 1 0 *	none	none	none
	EXTENDED READ INITIAL	X X	X^1 X	2 0 or 3 0 *	none	none	none
	READ	X X	X^1 X	0 1 or 1 1 *	none	none	none
	EXTENDED READ	X X	X^1 X	2 1 or 3 1 *	none	none	none

* Reading/writing is verified.

See: Disk Devices and Controls (Order No. 514)

Table 8-27 (cont). Summary of PDT I/O Control Characters

INPUT/OUTPUT OPERATION		PDT I/O CONTROL CHARACTER					
		C1 READ/WRITE CHANNEL	C2 CONTROL UNIT	C3 ADDITIONAL PARAMETERS	C4 ADDITIONAL PARAMETERS	C5 ADDITIONAL PARAMETERS	C6 ADDITIONAL PARAMETERS
CONSOLE	READ (NO CARRIAGE RETURN)	X X	X^1 X	0 0	none	none	none
	READ (CARRIAGE RETURN)	X X	X^1 X	0 1	none	none	none
	WRITE (NO CARRIAGE RETURN)	X X	X^2 X	0 0	none	none	none
	WRITE (CARRIAGE RETURN)	X X	X^2 X	0 1	none	none	none
See: Control Panels and Consoles (Models 200/1200/1250/2200), (Order No. 453)							
ON-LINE ADAPTER	TRANSFER ID character to Series 200 memory.	X X	X X	4 X (X=unused)	none	none	none
	ACCEPT the H-800/1800 instruction defined in the ID register.[7]	X X	X X	0 0	none	none	none
	ACCEPT the H-800/1800 instruction defined in the ID register, and cause the H-800/1800 to branch to U+3 or U+5.[7]	X X	X X	0 4	none	none	none
	DO NOT ACCEPT the H-800/1800 instruction defined in the ID register; rather, cause the H-800/1800 program to branch to U+6 or U+7 (read or write error).[7]	X X	X X	1 U (U = any value from 1 - 7, octal)	none	none	none
	SET the device busy indicator.[7]	X X	X X	3 X (X=unused)	none	none	none
See: Model 212 On-Line Adapter (DSI-274)							
TYPE 281 SCCC	RECEIVE	X X	X^1 X	none	none	none	none
	TRANSMIT	X X	X^2 X	none	none	none	none
TIME-OF-DAY CLOCK	TRANSFER TIME TO MEMORY	X X	X X	none	none	none	none
CENTRAL PROCESSOR ADAPTER	TRANSFER DATA	X X	X X	none	none	none	none
See: Type 212-1 Central Processor Adapter (Order No. 239)							

Table 8-27 (cont). Summary of PDT I/O Control Characters

INPUT/OUTPUT OPERATION		PDT I/O CONTROL CHARACTER					
		C1 READ/WRITE CHANNEL	C2 CONTROL UNIT	C3 ADDITIONAL PARAMETERS	C4 ADDITIONAL PARAMETERS	C5 ADDITIONAL PARAMETERS	C6 ADDITIONAL PARAMETERS
232 and 233-2 MICR READER-SORTERS	TRANSFER DATA	X X	X X	none	none	none	none
See: Type 233-2 MICR Control (Order No. 464)							
PLOTTER CONTROL	PLOT[8]	X X	X X	none	none	none	none
OPTICAL JOURNAL READER CONTROL	TRANSFER DATA	X X	X X	none	none	none	none
BILL FEED PRINTER CONTROL	PRINT	X X	X X	See Table 8-30	none	none	none
See: Type 237 Bill Feed Printer Control (Order No. 194)							

NOTES:
1. The high-order bit must be 1.
2. The high-order bit must be 0.
3. Odd parity is assumed. If even parity is required, the first octal character should be 7.
4. Odd parity is assumed. If even parity is required, the first octal character should be 3.
5. Odd parity and short gap are assumed. The first octal character should be 3 for even parity, short gap; 6 for odd parity, long gap; 7 for even parity, long gap.
6. D (tape drive) = 0-3 when the instruction is issued to the Type 203B-5 Tape Control. D = 0 or 1 when the instruction is issued to the Type 203C-7 Tape Control.
7. This operation issues initiating and concluding device-ready responses.
8. A complete plot can be executed by a single PDT instruction.

Table 8-28. C3 Coding for Type 209 and 209-2 Paper Tape Readers

VALUE	B BIT	A BIT	8 BIT	4 BIT	3 BIT	1 BIT
1	Not used	One character per frame	Sense end of record	Check odd parity	Read Forward	Increment CLC
0	Not used	Two characters per frame	Do not sense end of record	Check even parity	Read Reverse (Feature 010 or 011)	Decrement CLC

Table 8-29. C3 Coding for Type 210 Paper Tape Punch

VALUE	B BIT	A BIT	8 BIT	4 BIT	2 BIT	1 BIT
1	Not used	One character per frame	Not used	Compute odd parity	00 = Do not punch parity 01 = Parity bit in channel six 10 = Parity bit in channel seven 11 = Parity bit in channel eight	
0	Not used	Two characters per frame	Not used	Compute even parity		

Table 8-30. C3 Coding for Types 206 and 222 Printers and Type 237 Bill Feed Printer Control

Type 206 Printer		Type 222 Printers[1] and Type 237 Control	
C3	INTERPRETATION	C3	INTERPRETATION
00nnnn	Print, then space the number of lines specified by nnnn (1 - 15).	00nnnn	Print, then space the number of lines specified by nnnn (0 - 15).
01nnnn	Print, then space to the head of the form if the end of the form is sensed; otherwise, space the number of lines specified by nnnn (1 - 15).	01nnnn	Print, then space to channel one of the format tape (HOF) if channel two of the format tape (EOF) is sensed; otherwise, space the number of lines specified by nnnn (0 - 15).
11nnnn	Do not print; space the number of lines specified by nnnn (1 - 15).	11nnnn	Do not print; space the number of lines specified by nnnn (0 - 15).
100011	Print, then space to the head of the form.	100xxx 101xxx	Print, then space to channel xxx.[2] Do not print; space to channel xxx.[2]
101111	Do not print; space to the head of the form. 53	000 001 010 011 100 101 110 111	Channel 3 Channel 4 Channel 5 Channel 1 (Head of form) Channel 6 Channel 7 Channel 8 Channel 1 (Head of form)[3]

[1] Control characters are the same with or without the presence of the Print Buffer (Feature 036) in the printer.

[2] The basic Type 222-5 Printer can only space to channel 1; i.e., xxx must be 011 or 111. However, when equipped with Feature 1036 (8-channel Vertical Format Tape) this printer can space to any of the channels listed.

[3] In the 237 control, space to head of form if Read PDT has been received.

Table 8-31. C3 Coding for Type 270A Random Access Drum

VALUE	B BIT	A BIT	8 BIT	4 BIT	2 BIT	1 BIT
1	Override	Increment drum address register	This is a Read Address Register instruction	Drum file designation 0 - 7 (octal)		
0	Do not override	Do not increment drum address register	This is not a Read Address Register instruction			

Table 8-32. Summary of PDT I/O Control Characters for Type 286 Multi-Channel Communication Control

	INPUT/OUTPUT OPERATION	A ADDRESS	PDT I/O CONTROL CHAR.		
			C1	C2	C3
TYPE 286-1, -2, -3 MCCC	FIRST DATA TRANSMISSION PDT	LOC (specifies "line 0" in 286)	X X	X^1 X	none
	RECEIVE DATA PDT	LOC+2 (specifies line address in 286)	X X	X^1 X	none
	TRANSMIT DATA PDT	LOC+2 (specifies line address in 286)	X X	X^2 X	none
	LINE CONTROL PDT	LOC (specifies address of line to be controlled) NOTE: The line control transmission PDT instructions are listed in Table 8-33, below.	X X	X^2 X	none
TYPES 286-4 AND -5 MESSAGE-MODE MCCC	TRANSMIT (Load/test state only)	Leftmost character of field from which data is transferred.	X X	X^2 X	Section address or line number, 00_8 - 63_8.
	RECEIVE (Load/test state only)	Leftmost character of field to which data is transferred.	X X	X^1 X	Section address or line number, 00_8 - 63_8.
	ASSIGN RWC AND LOAD SLC (Initialized or off-line state only)	Leftmost character of 5-character status field storing interrupt information.	X X	X X	none

NOTES: 1. The high-order bit must be 1.
2. The high-order bit must be 0.

Table 8-33. Type 286-1, -2, -3 Line Control Instructions

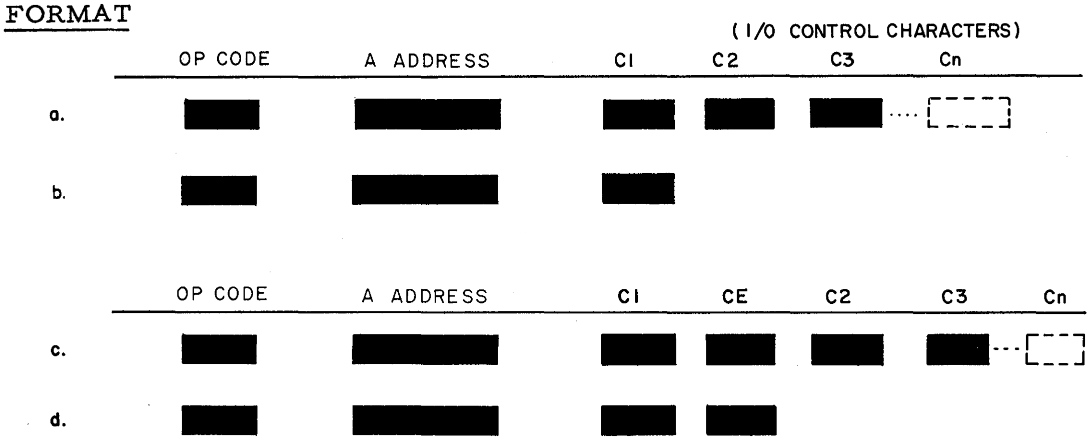
CODE[1] (OCTAL)	INSTRUCTION	DESCRIPTION
10	Transmit last character	Inform the 286 that the last character has been sent from the central processor, and place the control unit in the receive mode for that line (after transmitting last character).
60	Receive clear	Reset the bits of the logic character in the 286 memory. (This instruction should be given when power is first turned on.)

Table 8-33 (cont). Type 286-1, -2, -3 Line Control Instructions

CODE[1] (OCTAL)	INSTRUCTION	DESCRIPTION
30	Inhibit 285 (service request)	Turn off the interrupt capability of a line that is requesting service (either input or output).
50	Transmit idle character	Repeat the previously provided character indefinitely, without interrupts.
40	Transmit	Stop the line from repeating character and cause an interrupt.
74	Move Longitudinal Redundancy Check (LRC) Character	Move the LRC character from the LRC register to the data buffer register (Feature 087).
34	Special Strobe	Activate the special strobe line to a Type 285 adapter via the Type 286 control.
NOTE: The control code is stored in location LOC+1. (The low-order two bits of this code must be 0.)		

PCB PERIPHERAL CONTROL AND BRANCH

FORMAT



TYPES OF TEST AND CONTROL OPERATIONS

The Peripheral Control and Branch instruction can initiate four types of operations: (1) strictly mechanical peripheral device operations; (2) test and branch operations; (3) mode change operations; and (4) peripheral interrupt operations.

1. A mechanical operation is a non-data transfer operation such as rewind magnetic tape or seek a disk pack drive cylinder.
2. A test and branch operation tests the status of a peripheral control and/or a read/write channel(s). If the condition being tested (e.g., peripheral control busy, error in last card punched) is present, a program branch is performed.
3. A mode change operation conditions the addressed peripheral control to operate in a specific mode. For instance, the card reader control can be conditioned to reject illegally punched cards, to generate a busy signal if illegally punched cards are read, or both, depending upon the control characters of the PCB instruction.
4. A peripheral interrupt operation directs a peripheral control to change the setting of an interrupt function or an allow interrupt function (see Appendix D).

Control character C1 designates a read/write channel or combination of channels whose busy status is to be tested. If an RWC busy test is not desired, C1 must contain zeros. C2 designates the logical address of the peripheral control to be tested or actuated. The coding of this character is the same as its coding for a PDT instruction (see Table 8-26, page 8-118).

Control characters C3 through Cn designate the control and test operations. Any number of control characters may follow C2, each one designating a different operation. If control characters within a single instruction designate conflicting operations (e.g., punch Hollerith code and punch direct transcription mode), the control character to the left is cancelled by a conflicting control character to the right within the same instruction. If multiple test operations are specified within a single instruction, a branch will occur if any of the conditions tested is present. The specific use of characters C3 through Cn is dependent upon the type of peripheral device addressed. Tables 8-34 through 8-36 summarize the coding of these characters.

FUNCTION

Format a: The read/write channel or channel combination specified by C1 is tested for busy status. If it is busy, a branch is made to the instruction at A. If the RWC is not busy (or if C1 is 00_8), the operation(s) specified by characters C3 through Cn is performed on the peripheral control specified by C2. This peripheral control must be connected to the input/output sector implied by the value of C1.

Format b: The read/write channel or channel combination specified by C1 is tested for busy status. If it is busy, a branch is made to A. If the RWC is not busy, the instruction following the PCB is executed.

Format c: The read/write channel or channel combination specified by C1 is tested for busy status. Also, the sector designated by CE is interrogated to determine whether or not it has currently available sufficient unassigned memory accesses per unit time interval to support the I/O data transfer rate implied by C1, i.e., the data handling capacity of the RWC(s) designated by C1 (see Table 8-24). If the specified RWC is not busy and the designated sector can handle the data rate implied by C1, the operation(s) specified by characters C3 through Cn is performed on the peripheral control specified by C2, and the program continues in normal sequence. Otherwise, a branch is made to the instruction at A. The CE character must designate the I /O sector to which the peripheral control specified by C2 is connected (see Table 8-37 page 8-150).

Format d: The read/write channel or channel combination specified by C1 is tested for busy status. Also, the sector designated by CE is interrogated to determine whether or not it has currently available sufficient unassigned memory accesses per unit time interval to support the I/O data transfer rate implied by C1, i.e., the data handling capacity of the RWC(s) designated by C1 (see Table 8-24). If the specified RWC(s) is not busy and the designated sector can handle the data rate implied by C1, the program continues in sequence. Otherwise, a branch is made to the instruction at A.

PUNCTUATION MARKS

The execution of this instruction neither affects nor is affected by word marks or record marks.

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR	
NXT	A	B_p	NO BRANCH
JI (A)	A	NXT	BRANCH

NOTES

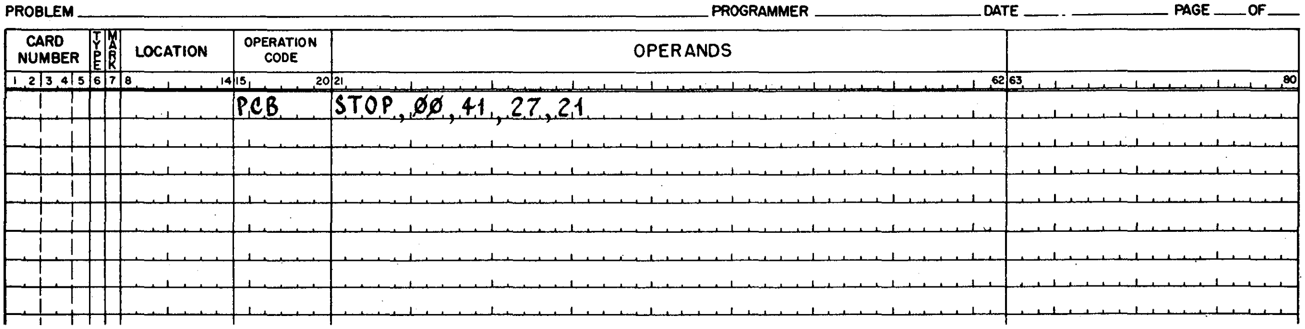
1. Formats c. and d. are applicable only to the Type 4201 processor. In order to produce a meaningful result, C1 must not be zero in these formats.
2. The PCB op code is a "privileged" op code when used in a Type 1201, 1251, 2201, or 4201 processor equipped with the Storage Protect Feature (see Appendix E).
3. Control character C1 of a PCB instruction is stored in the variant register.

EXAMPLE

In the following example, assume that the logical address of the card reader control is octal 41.

Set the card reader control to read Hollerith code (C3 = 27) and to reject automatically all cards with hole-count errors (C4 = 21). If the device is inoperable, branch to the location tagged STOP. (Note that since an RWC is not to be tested, C1 must contain zeros.)

EASYCODER
CODING FORM



PROBLEM ______ PROGRAMMER ______ DATE ___ . ______ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					PCB	STOP,ØØ,41,27,21	
2							
3							
4							
5							
6							
7							
8							

Table 8-34. Summary of PCB I/O Control Characters

OPERATION			PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
TYPE 214-1 CARD PUNCH	Branch to A address if device busy		X X	X X	1 0
	Branch to A address if punch-check error		X X	X X	4 1
	Branch to A address if device unavailable. If available, set control unit to:	Punch Hollerith code[4]	X X	X X	2 7
		Punch special code	X X	X X	2 6
		Punch direct transcription code (feature 064)	X X	X X	2 5
		Generate busy signal if punch-check error	X X	X X	2 3
		Offset-stack cards with punch-check error	X X	X X	2 1
		Offset-stack the card currently at the punch station	X X	X X	3 1
	Turn the control allow function OFF		X X	X X	7 0
	Turn the control allow function ON		X X	X X	7 1
	Turn the control interrupt function OFF		X X	X X	7 4
	Branch to A address if the control interrupt function is ON		X X	X X	7 5
See: Type 214-1 Card Punch (Order No. 451)					
TYPE 214-2 CARD READER/PUNCH	Branch to A address if device busy		X X	X[3] X	1 0
	Branch to A address if cycle-check or punch-check error		X X	X[3] X	4 1
	Branch to A address if illegal punch		X X	X[3] X	4 2
	Branch to A address if device unavailable. If available, set control unit to:	Terminate punch-feed read operations, operate in Hollerith mode, and accept all other cards[4]	X X	X[3] X	2 7
		Read or punch special code	X X	X[3] X	2 6

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION			PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
TYPE 214-2 CARD READER/PUNCH (cont)		Read or punch direct transcription code (feature 064)	X X	X[3] X	2 5
		Generate busy signal if illegal punch	X X	X[3] X	2 4
		Generate busy signal if cycle-check or punch-check error	X X	X[3] X	2 3
		Offset-stack cards with illegal punches	X X	X[3] X	2 2
		Offset-stack cards with cycle-check or punch-check error	X X	X[3] X	2 1
		Operate in punch-feed read mode	X X	X[3] X	2 0
		Offset-stack the card currently at the punch station	X X	X[3] X	3 1
	Turn the control allow function OFF		X X	X[3] X	7 0
	Turn the control allow function ON		X X	X[3] X	7 1
	Turn the control interrupt function OFF		X X	X[3] X	7 4
	Branch to A address if the control interrupt function is ON		X X	X[3] X	7 5
See: Type 214-2 Card Reader/Punch (Order No. 452)					
TYPE 223, 223-2 CARD READER	Branch to A address if device busy		X X	X X	1 0
	Branch to A address if cycle-check error		X X	X X	4 1
	Branch to A address if illegal punch		X X	X X	4 2
	Branch to A address if device unavailable. If available, set control unit to:	Read Hollerith code and accept all error cards[4]	X X	X X	2 7
		Read special code	X X	X X	2 6
		Read direct transcription code (feature 044)	X	X X	2 5
		Offset-stack cards with cycle-check error	X X	X X	2 1

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION			PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
TYPE 223, 223-2 CARD READER (cont)		Offset-stack cards with illegal punches	X X	X X	2 2
		Generate busy signal if cycle-check error	X X	X X	2 3
		Generate busy signal if illegal punch	X X	X X	2 4
		Offset-stack the card currently at the read station.	X X	X X	3 1
	Turn the control allow function OFF		X X	X X	7 0
	Turn the control allow function ON		X X	X X	7 1
	Turn the control interrupt function OFF		X X	X X	7 4
	Branch to A address if the control interrupt function is ON		X X	X X	7 5
See: Type 223 Card Reader (Order No. 504)					
TYPE 224-1, -2 CARD READER/CARD PUNCH	Branch to A address if device busy		X X	X[3] X	1 0
	Branch to A address if echo-check or read registration errors		X X	X[3] X	4 1
	Branch to A address if illegal punch		X X	X[3] X	4 2
	Branch to A address if device unavailable. If available, set control unit to:	Terminate punch-feed read operations, operate in Hollerith mode, and accept all error cards[4]	X X	X[3] X	2 7
		Convert to special code	X X	X[3] X	2 6
		Operate in direct transcription mode (Feature 064)	X X	X[3] X	2 5
		Generate busy signal if illegal punch	X X	X[3] X	2 4
		Generate busy signal if echo-check or read registration errors	X X	X[3] X	2 3
		Reject cards with illegal punches (Feature 065)	X X	X[3] X	2 2
		Reject cards with echo-check or read registration errors (Feature 065)	X X	X[3] X	2 1

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION			PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
TYPE 224-1, -2 CARD READER CARD PUNCH (cont)		Operate in punch-feed read mode	X X	X^3 X	2 0
		Reject card presently in the punch station	X X	X^3 X	3 1
	Turn the control allow function OFF		X X	X^3 X	7 0
	Turn the control allow function ON		X X	X^3 X	7 1
	Turn the control interrupt function OFF		X X	X^3 X	7 4
	Branch to A address if the control interrupt function is ON		X X	X^3 X	7 5
See: Type 224 Card Reader/Punch (Order No. 506)					
TYPE 227 CARD READER	Branch to A address if device busy		X X	X X	1 0
	Branch to A address if hole-count error		X X	X X	4 1
	Branch to A address if illegal punch		X X	X X	4 2
	Branch to A address if device unavailable. If available, set control unit to:	Terminate punch-feed read operations (Feature 062), if applicable, operate in Hollerith mode, and accept all error cards[4]	X X	X X	2 7
		Read special code	X X	X X	2 6
		Read direct transcription code (Feature 040)	X X	X X	2 5
		Reject cards with hole-count errors	X X	X X	2 1
		Reject cards with illegal punches	X X	X X	2 2
		Generate busy signal if hole-count error	X X	X X	2 3
		Generate busy signal if illegal punch	X X	X X	2 4
		Place previously read card in middle stacker (Feature 017)	X X	X X	3 1
		Place previously read card in the read eject stacker (Feature 017-1)	X X	X X	3 2

Table 8-34 (cont). Summary of PCB I/O Control Characters

	OPERATION		PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
TYPE 227 CARD READER (cont)	Turn the control allow function OFF		X X	X X	7 0
	Turn the control allow function ON		X X	X X	7 1
	Turn the control Interrupt function OFF		X X	X X	7 4
	Branch to A address if the control interrupt function is ON		X X	X X	7 5
See: Type 227 Card Reader/Punch (Order No. 564)					
TYPE 227 CARD PUNCH	Branch to A address if device busy		X X	X X	1 0
	Branch to A address if hole-count error (Feature 061)		X X	X X	4 1
	Branch to A address if device unavailable. If available, set control unit to:	Terminate punch-feed read operations (Feature 062), if applicable, and punch Hollerith code[4]	X X	X X	2 7
		Punch special code	X X	X X	2 6
		Punch direct transcription code (Feature 060)	X X	X X	2 5
		Reject cards with illegal punches (Feature 062)	X X	X X	2 2
		Reject cards with hole-count errors (Feature 061)	X X	X X	2 1
		Punch-feed read operations (Feature 062)	X X	X X	2 0
		Place previously punched card in middle stacker (Feature 017)	X X	X X	3 1
		Place previously punched card in the punch eject stacker (Feature 017-1)	X X	X X	3 2
	Turn the control allow function OFF		X X	X X	7 0
	Turn the control allow function ON		X X	X X	7 1

Table 8-34 (cont). Summary of PCB I/O Control Characters

	OPERATION		PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
TYPE 227 CARD PUNCH (cont)	Turn the control interrupt function OFF		X X	X X	7 4
	Branch to A address if the control interrupt function is ON		X X	X X	7 5
See: Type 227 Card Reader/Punch (Order No. 564)					
TYPE 209 and 209-2 PAPER TAPE READERS	Branch to A address if device busy		X X	X X	1 0
	Branch to A address if parity error		X X	X X	4 0
	Branch to A address if device unavailable. If available, set control unit to:	Rewind the tape (reverse direction)	X X	X X	3 0
		Run out the tape (forward direction)	X X	X X	3 2
	Turn the control allow function OFF		X X	X X	7 0
	Turn the control allow function ON		X X	X X	7 1
	Turn the control interrupt function OFF		X X	X X	7 4
	Branch to A address if the control interrupt function is ON		X X	X X	7 5
See: Types 209, 209-2, and 210 Paper Tape Equipment (Order No. 507)					
TYPE 210 PAPER TAPE PUNCH	Branch to A address if device busy		X X	X X	1 0
	Branch to A address if tape-low condition is true		X X	X X	6 0
	Turn the control allow function OFF		X X	X X	7 0
	Turn the control allow function ON		X X	X X	7 1
	Turn the control interrupt function OFF		X X	X X	7 4
	Branch to A address if the control interrupt function is ON		X X	X X	7 5
See: Types 209, 209-2, and 210 Paper Tape Equipment (Order No. 507)					
TYPE 206 PRINTER	Branch to A address if device busy		X X	X X	1 0
	Branch to A address if print error		X X	X X	4 0
See: Honeywell Series 200 Equipment Operators' Manual (Order No. 040)					

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION		PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
TYPE 222-1, -2, -3, -4, -5, -6 PRINTER	Branch to A address if device busy	X X	X X	1 0
	Branch to A address if print error	X X	X X	4 0
	Branch to A address if paper is moving	X X	X X	2 0
	Branch to A address if busy or paper is moving	X X	X X	3 0
	Branch to A address if end of form	X X	X X	0 1
	Branch to A address if channel eight	X X	X X	0 2
	Turn the control allow function OFF	X X	X X	7 0
	Turn the control allow function ON	X X	X X	7 1
	Turn the control interrupt function OFF	X X	X X	7 4
	Branch to A address if the control interrupt function is ON	X X	X X	7 5

See: Type 222 Printers (Order No. 562)

Notes: PCB instructions with C3 characters 01, 02, 20, and 30 are not applicable to the basic 222-5 printer. However, the 222-5 equipped with Feature 1036 (8-Channel Vertical Format Tape) can perform all of the operations listed.

Control characters are the same with or without the presence of the Print Buffer (Feature 036) in the printer.

OPERATION		PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
MAGNETIC TAPE UNITS 1/2-INCH	Rewind	X X	X^2 X	2 D (D=tape drive, 0 - 7)
	Rewind and release	X X	X^1 X	2 D (D=tape drive, 0 - 7)
	Branch to A address if read busy	X X	X^1 X	0 D (D=tape drive, 0 - 7)
	Branch to A address if write busy	X X	X^2 X	0 D (D=tape drive, 0 - 7)
	Branch to A address if read/write error	X X	X^2 X	4 D (D=tape drive, 0 - 7)
	Branch to A address if beginning of tape	X X	X^1 X	6 D (D=tape drive, 0 - 7)

Table 8-34 (cont). Summary of PCB I/O Control Characters

	OPERATION	PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
MAGNETIC TAPE UNITS 1/2-INCH (cont)	Branch to A address is end of tape	X X	X^2 X	6 D (D=tape drive, 0 - 7)
	Turn the control allow function OFF	X X	X^3 X	7 0
	Turn the control allow function ON	X X	X^3 X	7 1
	Turn the control interrupt function OFF	X X	X^3 X	7 4
	Branch to A address if the control interrupt function is ON	X X	X^3 X	7 5

See: Type 204B Series Magnetic Tape Units (Order No. 503), Types 204B-11 and 204B-12 Magnetic Tape Units (Order No. 502), or Types 204C-13 and 204C-14 Magnetic Tape Units (Order No. 623)

Note: The Type 204B-11 and 204B-12 Magnetic Tape Units are limited to tape drive designations in the range 0-3 and are unable to execute the command "rewind and release." Tape drive designations for the Type 204C-13 and 204C-14 Magnetic Tape Units must be either 0 or 1; these units cannot execute the "rewind and release" command.

	OPERATION	C1	C2	C3 through Cn
MAGNETIC TAPE UNITS 3/4-INCH	Rewind	X X	X^2 X	2 D (D=tape drive, 0 - 3)
	Release	X X	X^1 X	2 D (D=tape drive, 0 - 3)
	Branch to A address if read busy	X X	X^1 X	0 D (D=tape drive, 0 - 3)
	Branch to A address if write busy	X X	X^2 X	0 D (D=tape drive, 0 - 3)
	Branch to A address if read/write error	X X	X X	4 D (D=tape drive, 0 - 3)
	Branch to A address if beginning of tape	X X	X^1 X	6 D (D=tape drive, 0 - 3)
	Branch to A address if end of tape	X X	X^2 X	6 D (D=tape drive, 0 - 3)
	Branch to A address if "long check" error is detected	X X	X^2 X	5 X (X=unused)

Table 8-34 (cont). Summary of PCB I/O Control Characters

	OPERATION	PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
MAGNETIC TAPE UNITS 3/4-INCH (cont)	Turn the control allow function OFF	X X	X^3 X	7 0
	Turn the control allow function ON	X X	X^3 X	7 1
	Turn the control interrupt function OFF	X X	X^3 X	7 4
	Branch to A address if the control interrupt function is ON	X X	X^3 X	7 5
See:	Series 200 Equipment Operators' Manual (Order No. 040), Type 204A Series Magnetic Tape Units (Order No. 863)			
TYPE 270A RANDOM ACCESS DRUM	Branch to A address if device busy[5]	X X	X X	0 X or 1 X (X=unused)
	Branch to A address if error indicator is ON	X X	X X	4 X (X=unused)
	Turn the control allow function OFF	X X	X X	7 0
	Turn the control allow function ON	X X	X X	7 1
	Turn the control interrupt function OFF	X X	X X	7 4
	Branch to A address if the control interrupt function is ON	X X	X X	7 5
See:	Type 270A Random Access Drum and Control (Order No. 009)			
TYPE 220-1 CONSOLE	Branch to A address if device busy	X X	X^2 X	1 0
See:	Control Panels and Consoles (Models 200/1200/1250/2200), (Order No. 453)			
TYPE 220-2 CONSOLE	Branch to A address if device busy	X X	X^2 X	1 0
	Reset the interrupt function	X X	X^2 X	7 6
	Branch to A address if the interrupt function is ON	X X	X^2 X	7 7
See:	Control Panels and Consoles (Models 200/1200/1250/2200), (Order No. 453)			

Table 8-34 (cont). Summary of PCB I/O Control Characters

	OPERATION	PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
TYPE 220-3 CONSOLE	Branch to A address if device busy	X X	X^2 X	1 0
	Turn the allow function OFF	X X	X^2 X	7 0
	Turn the allow function ON[8]	X X	X^2 X	7 1
	Turn the data termination interrupt function OFF	X X	X^2 X	7 4
	Branch to A address if data termination interrupt function is ON	X X	X^2 X	7 5
	Turn the manual interrupt function OFF[9]	X X	X^2 X	7 6
	Branch to A address if manual interrupt function is ON[9]	X X	X^2 X	7 7
See: Control Panels and Consoles (Models 200/1200/1250/2200), (Order No. 453)				
TYPE 212 ON-LINE ADAPTER	Branch to A address if device busy	X X	X^3 X	0 X or 1 X (X=unused)
	Branch to A address if data transfer is in progress	X X	X^3 X	7 X (X=unused)
	Branch to A address if error or incomplete indicator is set	X X	X^3 X	4 X
	Branch to A address if parity error is stored	X X	X^3 X	5 X (X=unused)
	Branch to A address if incomplete error is stored	X X	X^3 X	6 X (X=unused)
	Place control character C4 in the ID register if data transfer is not in progress	X X	X^3 X	C3: 2 X (X=unused) C4: octal character to be placed in ID register
	Branch to A address unconditionally, and clear the ID register	X X	X^3 X	3 X (X=unused)
See: Model 212 On-Line Adapter (DSI-274)				
DISK DE-VICES	Branch to A address if specified device is busy; otherwise, set control unit to: Seek out the cylinder (specified by C5 and C6) in the pack (specified by C4).	X X	X^2 X	C3: 2 D (D=device address, 0-7) C4: 00

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION			PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
DISK DEVICES (cont)					C5 and C6: 0000 to 0143 for the Type 258, 0000 to 0312 for the Type 259.
		Restore the specified device to cylinder zero.	X X	X[1] X	3D (D=device address, 0-7).
	Branch to A address if specified device is not busy; otherwise, set control unit to:	Continue with the next sequential record the operation (read or write) being performed with the current record.	X X	X[1] X	7D (D=device address, 0-7).
	Branch to A address if control busy.		X X	X[2] X	1 0
	Branch to A address if device busy.		X X	X[2] X	C3: 0 D (D=device address, 0-7) C4: 0 0 or another valid C3 character
	Branch to A address if a general exception condition occurred during the preceding PDT instruction.		X X	X[2] X	5 0
	Branch to A address if the TLR flag is set.		X X	X[2] X	6 0
	Set control unit to override setting of FORMAT WRITE PERMIT switch.		X X	X[2] X	4 0
	Turn control allow function OFF.		X X	X[2] X	7 0
	Turn control allow function ON.		X X	X[2] X	7 1
	Turn drive allow function OFF.		X X	X[2] X	7 2
	Turn drive allow function ON.		X X	X[2] X	7 3
	Turn control interrupt function OFF.[7]		X X	X[2] X	7 4
	Branch to A address if control interrupt function is ON.		X X	X[2] X	7 5
	Turn drive interrupt function OFF.		X X	X[2] X	7 6

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION		PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
DISK DE-VICES	Branch to A address if device interrupt function is ON.	X X	X^2 X	7 7
See: Disk Devices and Controls (Order No. 514)				
TYPE 281 SINGLE CHANNEL COMMUNICATION CONTROL	Branch to A address if device busy	X X	X^3 X	1 0
	Branch to A address if parity error	X X	X^3 X	4 0
	Branch to A address if error other than parity error	X X	X^3 X	5 0
	Branch to A address if the 281 is in transmit mode and requesting data for transmission onto line	X X	X^3 X	6 0
	Branch to A address if the 281 is in receive mode and requesting that central processor take received data	X X	X^3 X	6 1
	Turn the allow function OFF	X X	X^6 X	7 0
	Turn the allow function ON	X X	X^6 X	7 1
	Turn the interrupt function OFF	X X	X^6 X	7 4
	Branch to A address if allow and interrupt functions are ON	X X	X^6 X	7 5
TYPE 213-3 INTERVAL TIMER	Branch to A address if device busy (Feature 071)	0 0	X^6 X	1 0
	Turn the allow function OFF	0 0	X^6 X	7 0
	Turn the allow function ON	0 0	X^6 X	7 1
	Turn the allow function ON (Feature 071)	0 0	X^6 X	7 3 (C4 - C6 specify time interval)
	Turn the interrupt function OFF	0 0	X^6 X	7 4
	Branch to A address if interrupt function is ON	0 0	X^6 X	7 5
	Turn the interrupt function OFF (Feature 071)	0 0	X^6 X	7 6
	Branch to A address if interrupt function is ON (Feature 071)	0 0	X^6 X	7 7
See: Type 213-3 Interval Timer and Feature 071 Interval Selector (Order No. 082)				

Table 8-34 (cont). Summary of PCB I/O Control Characters

	OPERATION	PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
TYPE 213-4 TIME OF DAY CLOCK	Branch to A address if device busy	X X	X X	1 0
TYPE 212-1 CENTRAL PROCESSOR ADAPTER	Branch to A address if device busy	X X	X^6 X	0 X or 1 0
	Branch to A address if device busy, and reserve	X X	X^6 X	C3: 2 0 C4: 0 0
	Branch to A address if reserve action by this central processor was not successful	X X	X^6 X	C3: 2 0 C4: 0 0 C5: 6 1
	Branch to A address if 212-1 is not set for data transfer (initiator)	X X	X^6 X	6 1
	Branch to A address if 212-1 is set for data transfer (responder)	X X	X^6 X	6 4
	Turn the allow function OFF	X X	X^6 X	7 0
	Turn the allow function ON	X X	X^6 X	7 1
	Turn the interrupt function OFF	X X	X^6 X	7 4
	Branch to A address if allow and interrupt functions are ON	X X	X^6 X	7 5
See: Type 212-1 Central Processor Adapter (Order No. 239)				
TYPE 234 PLOTTER CONTROL	Branch to A address if control unit busy.	X X	X X	1 0
	Turn the allow function OFF	X X	X X	7 0
	Turn the allow function ON	X X	X X	7 1
	Turn the interrupt function OFF	X X	X X	7 4
	Branch to A address if interrupt function is ON	X X	X X	7 5
See: Type 234 Plotter Control (Order No. 561)				

Table 8-34 (cont). Summary of PCB I/O Control Characters

	OPERATION		PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
TYPE 235 OPTICAL JOURNAL READER CONTROL	Branch to A address if device busy		X X	X X	1 0
	Branch to A address if reader is not set for data transfer or if control is busy		X X	X X	0 1
	Turn the allow function OFF		X X	X X	7 0
	Turn the allow function ON		X X	X X	7 1
	Turn the interrupt function OFF		X X	X X	7 4
	Branch to A address if interrupt function is ON		X X	X X	7 5
TYPE 232 and 233-2 MICR READER-SORTERS	Branch to A address if control busy		X X	X X	1 0
	Select stacker designated; Branch to A address if: 1. the reader-sorter is not ready; or 2. the 10-millisecond stacker selection period has elapsed; or 3. the leading edge of the document to be sorted has not passed the reading station; or 4. the leading edge has passed the reading station and a PDT instruction has not yet been issued; or 5. the reader-sorter is performing an automatic reject on the document in question	Stacker 0	X X	X X	2 0
		Stacker 1	X X	X X	2 1
		Stacker 2	X X	X X	2 2
		Stacker 3	X X	X X	2 3
		Stacker 4	X X	X X	2 4
		Stacker 5	X X	X X	2 5
		Stacker 6	X X	X X	2 6
		Stacker 7	X X	X X	2 7
		Stacker 8	X X	X X	3 0
		Stacker 9	X X	X X	3 1
		Stacker X	X X	X X	3 2
		Stacker Y	X X	X X	3 3
		Reject Stacker	X X	X X	3 7
	Start feed. Branch to A address if feed cannot be started due to: 1. the reader-sorter not being ready; or 2. proper restart procedures not followed		X X	X X	3 4
	Stop feed. Branch to A address if sorter-reader is not ready		X X	X X	3 5
	Set pocket-light control. Branch to A address if:		X X	X X	3 6

Table 8-34 (cont). Summary of PCB I/O Control Characters

OPERATION			PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
TYPE 232 and 233-2 MICR READER-SORTERS (cont)	1. the reader-sorter is not ready; or 2. a pocket-light control PCB is already in process				
	Branch to A address if:	Amount field error	X X	X X	4 0
		Process control field error	X X	X X	4 1
		Account field error	X X	X X	4 2
		Transit field error	X X	X X	4 3
		Auxiliary on-us field error	X X	X X	4 4
		Device error	X X	X X	5 0
		Passed document condition	X X	X X	5 1
	Operate in normal mode		X X	X X	6 0
	Operate in short-document mode		X X	X X	6 1
	Branch to A address if on-us field is complete		X X	X X	6 2
	Branch to A address if last document was a control document		X X	X X	6 3
	Branch to A address if end-of-file		X X	X X	6 4
	Advance batch counter one digit. Branch to A address if the sorter-reader is not stopped or the batch counter is currently being advanced.		X X	X X	6 5
	Turn allow function OFF		X X	X X	7 0
	Turn allow function ON		X X	X X	7 1
	Turn interrupt function OFF		X X	X X	7 4
	Branch to A address if interrupt function is ON		X X	X X	7 5

See: Type 233-2 MICR Control (Order No. 464)

Table 8-34 (cont). Summary of PCB I/O Control Characters

	OPERATION		PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
TYPE 237 BILL FEED PRINTER CONTROL	Branch to A address if:	Device busy	X X	X[3] X	1 0
		Form is moving	X X	X[2] X	2 0
		Device busy or form is moving	X X	X[3] X	3 0
		Print error or read check	X X	X[3] X	4 0
		Validity error	X X	X[3] X	4 1
	Branch on channel 2 (EOF) of format tape		X X	X[2] X	0 1
	Branch on channel 8 of format tape		X X	X[2] X	0 2
	Turn on validity check indicator		X X	X[1] X	2 0
	Turn the allow function OFF		X X	X X	7 0
	Turn the allow function ON		X X	X X	7 1
	Turn the interrupt function OFF		X X	X X	7 4
	Branch to A address if read interrupt function is ON		X X	X[1] X	7 5

NOTE: The two operations "Branch to A address if form is moving" and "Turn on validity check indicator" are both specified with a C3 character of 20_8, but are distinguished by the high-order bit of C2.

See: Type 237 Bill Feed Printer Control (Order No. 194)

NOTES:

1. The high-order bit must be 1.
2. The high-order bit must be 0.
3. The high-order bit is set to 1 for input operations and to 0 for output operations.
4. This control character should precede all other control characters that set the control to perform a certain action. It is the programmer's responsibility to set the control to the desired mode of operation at the beginning of the run.
5. As the drum control does not permit reading from one drum file while writing on another, it is considered busy if either a read or a write operation is in progress. (The value of the high-order bit in C2 is thus immaterial in this case.)
6. The high-order bit is ignored.
7. The interrupt functions of both the control and the disk device are automatically turned on when a "not busy" status is reached by the control or the disk device, respectively.
8. For program interruption in the 201-0 central processor, the processor must contain the Program Interrupt Feature (012).
9. The manual interrupt function is applicable only in those cases where the Type 220-3 is employed with the 201-0 or 201-1 central processpr; C3 control characters 76 and 77 perform no operations with other central processors. In those cases where the 201-0 or 201-1 is not equipped with the Program Interrupt Feature (012), the manual interrupt function can still be tested or turned off. Thus although the interrupt button cannot effect a manual interrupt, the corresponding function can be tested to set up a programmed interrupt.

Table 8-35. Summary of PCB I/O Control Characters for Type 286 Multi-Channel Communication Control

	OPERATION	PCB I/O CONTROL CHARACTERS			
		C1	C2	C3	C4 through Cn
TYPE 286-1, -2 -3 MCCU	Branch to A address if device busy. If not busy, set the 286 to stop scanning and continue the program in sequence	X X	X X	1 0	none
	Turn the allow function OFF	X X	X X	7 0	none
	Turn the allow function ON	X X	X X	7 1	none
	Branch to A address if the interrupt was due to the 286 requesting service	X X	X X	7 5	none
TYPE 286-4, -5 MESSAGE-MODE MCCU	Branch to A address if device busy	X X	X X	1 0	none
	Branch to A address if parity error	X X	X X	4 0	none
	Branch to A address if the interrupt was due to the 286 requesting service	X X	X X	7 5	none
	Turn the allow function ON	X X	X X	7 1	none
	Turn the allow function OFF	X X	X X	7 0	none
	Set the 286 to the load/test state	X X	X X	2 5	none
	Provide line orientation for load/test operation	X X	X X	4 1	none
	Turn the load/test state and line orientation OFF	X X	X X	2 4	none
	Turn the interrupt function OFF	X X	X X	7 4	none
	Release the RWC(S) assigned to the 286	X X	X X	2 7	none
	Set the halt/continue indicator to halt	X X	X X	2 0	none
	Set the halt/continue indicator to continue	X X	X X	2 1	none
	Turn the parity error indicator and the parity error interrupt function OFF	X X	X X	2 6	none
	Request the address of the next transfer that is to take place from the line designated by C4, and branch to the A address	X X	X X	3 6	C4: 00 to 77
	Abort the present instruction to the line designated by C4, generate an interrupt, initiate the next instruction to the same line, and branch to the A address	X X	X X	3 3	C4: 00 to 77
	Abort the present instruction to the line designated be C4, initiate the next instruction to the same line, and branch to the A address	X X	X X	3 2	C4: 00 to 77
	Reset synchronization for the line designated by C4, and branch to the A address	X X	X X	3 7	C4: 00 to 77
	Activate the special strobe line to the 285 adapter designated by C4, and branch to the A address	X X	X X	3 4	C4: 00 to 77
	Deliver to the 286 the information specified by C5 et seq. for the next instruction to the line designated by C4, and branch to the A address	X X	X X	3 0	C4: 00 to 77 (See Table 8-36 for C5 et seq.)

Table 8-35 (cont). Summary of PCB I/O Control Characters for Type 286 Multi-Channel Communication Control

	OPERATION	PCB I/O CONTROL CHARACTERS			
		C1	C2	C3	C4 through Cn
TYPE 286-4, -5 MESSAGE-MODE MCCU (cont)	Deliver to the 286 the information specified by C5 et seq. for the next instruction to the line designated by C4; then abort the present instruction to that line, generate an interrupt, initiate the next instruction to the same line, and branch to the A address	X X	X X	3 3	C4: 00 to 77 (See Table 8-36 for C5 et seq.)
	Deliver to the 286 the information specified by C5 et seq. for the next instruction to the line designated by C4, then abort the present instruction to that line, initiate the next instruction to the same line, and branch to the A address	X X	X X	3 2	C4: 00 to 77 (See Table 8-36 for C5 et seq.)

Table 8-36. PCB Control Characters C5 through C15 for Type 286-4, -5 Line Control Instructions

Control Character	Configuration (Octal)		Description
C5 C6 C7	C5 C6 C7 $\underbrace{\text{XX}}_{\text{most significant six bits}}$ $\underbrace{\text{XX}}_{\text{middle six bits}}$ $\underbrace{\text{XX}}_{\text{least significant six bits}}$		Address to be loaded into RWC counters (SLC and CLC) prior to data transfer.
C8 C9	C8 XX Bits 6 and 5 specify mode of operation of the line: 6 5 0 0 - Inhibit 0 1 - Receive 1 0 - Transmit 1 1 - Transmit Repeat	C9 XX Bit 6 is the response bit 0 - no interrupt is allowed at termination of the instruction. 1 - an interrupt is allowed.	Control characters which specify line action; they are loaded into the next instruction section of memory.
	Bit 4 is the Allow Timer bit 0 - Timer is not allowed 1 - Timer is allowed	Bits 4 and 5 are not used and must be zero.	

Table 8-36 (cont). PCB Control Characters C5 through C15 for Type 286-4, -5 Line Control Instructions

Control Character	Configuration (Octal)		Description
C8 C9 (cont)	Bits 3 and 2 specify character parity 3 2 0 0 no parity 0 1 generation or checking is performed 1 0 even parity 1 1 odd parity	Bit 3 is the block parity bit 0 - block parity is not used 1 - block parity is used	
	Bit 1 is the character transfer bit 0 - one six-bit character transfer per line character 1 - two six-bit character transfers per line character	Bit 2 is the command termination bit 0 - character recognized is the last one transferred 1 - one more data transfer is made to or from the CP after the character recognized and before command termination. Bit 1 defines block parity check bit 0 - check bit will be the half add sum of the parity bit of the preceding characters in the message. 1 - block parity character will have same parity generated or checked as the data characters.	
C10 C11	C10 XX	C11 XX	Eight bits (the low-order two bits of C10 and all six bits of C11) contain the first recognition character.
C12 C13	C12 XX	C13 XX	Eight bits (the low-order two bits of C12 and all six bits of C13) contain the second recognition character.

Table 8-36 (cont). PCB Control Characters C5 through C15 for Type 286-4, -5 Line Control Instructions

Control Character	Configuration (Octal)	Description
C14 C15	C14 C15 XX XX	Eight bits (the low-order two bits of C14 and all six bits of C15) contain the SIT character for asynchronous lines.

Table 8-37. Description of PCB I/O Character CE

Escape Code	Sector to Which Device is Connected
10	Sector 1
12	Sector 2
13	Sector 3

APPENDIX A

OCTAL NOTATION

Octal notation is a convenient shorthand method of writing pure binary numbers. In Series 200 programming it is used to represent such binary values as main memory addresses, variant characters, I/O control characters, and constants.

If a binary value is divided into groups of three bits, proceeding from right to left, each group may be replaced by its octal equivalent as indicated in Table A-1.

Table A-1. Binary-Octal Equivalents

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

Example 1.

The binary value

011111000101001110

when divided into three-bit groups

011 111 000 101 001 110

has an octal equivalent of

3 7 0 5 1 6

Example 2.

The binary value

1010100111010

when divided into three-bit groups

1 010 100 111 010

has an octal equivalent of

1 2 4 7 2

Table A-2. Decimal-Octal Conversion Table

LOW-ORDER OCTAL DIGIT	DECIMAL INCREMENT																									LOW-ORDER OCTAL DIGIT
0	000	008	016	024	032	040	048	056	064	072	080	088	096	104	112	120	128	136	144	152	160	168	176	184	192	0
1	001	009	017	025	033	041	049	057	065	073	081	089	097	105	113	121	129	137	145	153	161	169	177	185	193	1
2	002	010	018	026	034	042	050	058	066	074	082	090	098	106	114	122	130	138	146	154	162	170	178	186	194	2
3	003	011	019	027	035	043	051	059	067	075	083	091	099	107	115	123	131	139	147	155	163	171	179	187	195	3
4	004	012	020	028	036	044	052	060	068	076	084	092	100	108	116	124	132	140	148	156	164	172	180	188	196	4
5	005	013	021	029	037	045	053	061	069	077	085	093	101	109	117	125	133	141	149	157	165	173	181	189	197	5
6	006	014	022	030	038	046	054	062	070	078	086	094	102	110	118	126	134	142	150	158	166	174	182	190	198	6
7	007	015	023	031	039	047	055	063	071	079	087	095	103	111	119	127	135	143	151	159	167	175	183	191	199	7

DECIMAL BASE NO.																										DECIMAL BASE NO.
0000	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20	21	22	23	24	25	26	27	30	0000
0200	31	32	33	34	35	36	37	40	41	42	43	44	45	46	47	50	51	52	53	54	55	56	57	60	61	0200
0400	62	63	64	65	66	67	70	71	72	73	74	75	76	77	100	101	102	103	104	105	106	107	110	111	112	0400
0600	113	114	115	116	117	120	121	122	123	124	125	126	127	130	131	132	133	134	135	136	137	140	141	142	143	0600
0800	144	145	146	147	150	151	152	153	154	155	156	157	160	161	162	163	164	165	166	167	170	171	172	173	174	0800
1000	175	176	177	200	201	202	203	204	205	206	207	210	211	212	213	214	215	216	217	220	221	222	223	224	225	1000
1200	226	227	230	231	232	233	234	235	236	237	240	241	242	243	244	245	246	247	250	251	252	253	254	255	256	1200
1400	257	260	261	262	263	264	265	266	267	270	271	272	273	274	275	276	277	300	301	302	303	304	305	306	307	1400
1600	310	311	312	313	314	315	316	317	320	321	322	323	324	325	326	327	330	331	332	333	334	335	336	337	340	1600
1800	341	342	343	344	345	346	347	350	351	352	353	354	355	356	357	360	361	362	363	364	365	366	367	370	371	1800
2000	372	373	374	375	376	377	400	401	402	403	404	405	406	407	410	411	412	413	414	415	416	417	420	421	422	2000
2200	423	424	425	426	427	430	431	432	433	434	435	436	437	440	441	442	443	444	445	446	447	450	451	452	453	2200
2400	454	455	456	457	460	461	462	463	464	465	466	467	470	471	472	473	474	475	476	477	500	501	502	503	504	2400
2600	505	506	507	510	511	512	513	514	515	516	517	520	521	522	523	524	525	526	527	530	531	532	533	534	535	2600
2800	536	537	540	541	542	543	544	545	546	547	550	551	552	553	554	555	556	557	560	561	562	563	564	565	566	2800
3000	567	570	571	572	573	574	575	576	577	600	601	602	603	604	605	606	607	610	611	612	613	614	615	616	617	3000
3200	620	621	622	623	624	625	626	627	630	631	632	633	634	635	636	637	640	641	642	643	644	645	646	647	650	3200
3400	651	652	653	654	655	656	657	660	661	662	663	664	665	666	667	670	671	672	673	674	675	676	677	700	701	3400
3600	702	703	704	705	706	707	710	711	712	713	714	715	716	717	720	721	722	723	724	725	726	727	730	731	732	3600
3800	733	734	735	736	737	740	741	742	743	744	745	746	747	750	751	752	753	754	755	756	757	760	761	762	763	3800
4000	764	765	766	767	770	771	772	773	774	775	776	777	1000	1001	1002	1003	1004	1005	1006	1007	1010	1011	1012	1013	1014	4000
4200	1015	1016	1017	1020	1021	1022	1023	1024	1025	1026	1027	1030	1031	1032	1033	1034	1035	1036	1037	1040	1041	1042	1043	1044	1045	4200
4400	1046	1047	1050	1051	1052	1053	1054	1055	1056	1057	1060	1061	1062	1063	1064	1065	1066	1067	1070	1071	1072	1073	1074	1075	1076	4400
4600	1077	1100	1101	1102	1103	1104	1105	1106	1107	1110	1111	1112	1113	1114	1115	1116	1117	1120	1121	1122	1123	1124	1125	1126	1127	4600
4800	1130	1131	1132	1133	1134	1135	1136	1137	1140	1141	1142	1143	1144	1145	1146	1147	1150	1151	1152	1153	1154	1155	1156	1157	1160	4800
5000	1161	1162	1163	1164	1165	1166	1167	1170	1171	1172	1173	1174	1175	1176	1177	1200	1201	1202	1203	1204	1205	1206	1207	1210	1211	5000
5200	1212	1213	1214	1215	1216	1217	1220	1221	1222	1223	1224	1225	1226	1227	1230	1231	1232	1233	1234	1235	1236	1237	1240	1241	1242	5200
5400	1243	1244	1245	1246	1247	1250	1251	1252	1253	1254	1255	1256	1257	1260	1261	1262	1263	1264	1265	1266	1267	1270	1271	1272	1273	5400
5600	1274	1275	1276	1277	1300	1301	1302	1303	1304	1305	1306	1307	1310	1311	1312	1313	1314	1315	1316	1317	1320	1321	1322	1323	1324	5600
5800	1325	1326	1327	1330	1331	1332	1333	1334	1335	1336	1337	1340	1341	1342	1343	1344	1345	1346	1347	1350	1351	1352	1353	1354	1355	5800
6000	1356	1357	1360	1361	1362	1363	1364	1365	1366	1367	1370	1371	1372	1373	1374	1375	1376	1377	1400	1401	1402	1403	1404	1405	1406	6000
6200	1407	1410	1411	1412	1413	1414	1415	1416	1417	1420	1421	1422	1423	1424	1425	1426	1427	1430	1431	1432	1433	1434	1435	1436	1437	6200
6400	1440	1441	1442	1443	1444	1445	1446	1447	1450	1451	1452	1453	1454	1455	1456	1457	1460	1461	1462	1463	1464	1465	1466	1467	1470	6400
6600	1471	1472	1473	1474	1475	1476	1477	1500	1501	1502	1503	1504	1505	1506	1507	1510	1511	1512	1513	1514	1515	1516	1517	1520	1521	6600
6800	1522	1523	1524	1525	1526	1527	1530	1531	1532	1533	1534	1535	1536	1537	1540	1541	1542	1543	1544	1545	1546	1547	1550	1551	1552	6800
7000	1553	1554	1555	1556	1557	1560	1561	1562	1563	1564	1565	1566	1567	1570	1571	1572	1573	1574	1575	1576	1577	1600	1601	1602	1603	7000
7200	1604	1605	1606	1607	1610	1611	1612	1613	1614	1615	1616	1617	1620	1621	1622	1623	1624	1625	1626	1627	1630	1631	1632	1633	1634	7200
7400	1635	1636	1637	1640	1641	1642	1643	1644	1645	1646	1647	1650	1651	1652	1653	1654	1655	1656	1657	1660	1661	1662	1663	1664	1665	7400
7600	1666	1667	1670	1671	1672	1673	1674	1675	1676	1677	1700	1701	1702	1703	1704	1705	1706	1707	1740	1711	1712	1713	1714	1715	1716	7600
7800	1717	1720	1721	1722	1723	1724	1725	1726	1727	1730	1731	1732	1733	1734	1735	1736	1737	1740	1741	1742	1743	1744	1745	1746	1747	7800
8000	1750	1751	1752	1753	1754	1755	1756	1757	1760	1761	1762	1763	1764	1765	1766	1767	1770	1771	1772	1773	1774	1775	1776	1777	2000	8000
8200	2001	2002	2003	2004	2005	2006	2007	2010	2011	2012	2013	2014	2015	2016	2017	2020	2021	2022	2023	2024	2025	2026	2027	2030	2031	8200
8400	2032	2033	2034	2035	2036	2037	2040	2041	2042	2043	2044	2045	2046	2047	2050	2051	2052	2053	2054	2055	2056	2057	2060	2061	2062	8400
8600	2063	2064	2065	2066	2067	2070	2071	2072	2073	2074	2075	2076	2077	2100	2101	2102	2103	2104	2105	2106	2107	2110	2111	2112	2113	8600
8800	2114	2115	2116	2117	2120	2121	2122	2123	2124	2125	2126	2127	2130	2131	2132	2133	2134	2135	2136	2137	2140	2141	2142	2143	2144	8800
9000	2145	2146	2147	2150	2151	2152	2153	2154	2155	2156	2157	2160	2161	2162	2163	2164	2165	2166	2167	2170	2171	2172	2173	2174	2175	9000
9200	2176	2177	2200	2201	2202	2203	2204	2205	2206	2207	2210	2211	2212	2213	2214	2215	2216	2217	2220	2221	2222	2223	2224	2225	2226	9200
9400	2227	2230	2231	2232	2233	2234	2235	2236	2237	2240	2241	2242	2243	2244	2245	2246	2247	2250	2251	2252	2253	2254	2255	2256	2257	9400
9600	2260	2261	2262	2263	2264	2265	2266	2267	2270	2271	2272	2273	2274	2275	2276	2277	2300	2301	2302	2303	2304	2305	2306	2307	2310	9600
9800	2311	2312	2313	2314	2315	2316	2317	2320	2321	2322	2323	2324	2325	2326	2327	2330	2331	2332	2333	2334	2335	2336	2337	2340	2341	9800
10,000	2342	2343	2344	2345	2346	2347	2350	2351	2352	2353	2354	2355	2356	2357	2360	2361	2362	2363	2364	2365	2366	2367	2370	2371	2372	10,000
10,200	2373	2374	2375	2376	2377	2400	2401	2402	2403	2404	2405	2406	2407	2410	2411	2412	2413	2414	2415	2416	2417	2420	2421	2422	2423	10,200
10,400	2424	2425	2426	2427	2430	2431	2432	2433	2434	2435	2436	2437	2440	2441	2442	2443	2444	2445	2446	2447	2450	2451	2452	2453	2454	10,400
10,600	2455	2456	2457	2460	2461	2462	2463	2464	2465	2466	2467	2470	2471	2472	2473	2474	2475	2476	2477	2500	2501	2502	2503	2504	2505	10,600
10,800	2506	2507	2510	2511	2512	2513	2514	2515	2516	2517	2520	2521	2522	2523	2524	2525	2526	2527	2530	2531	2532	2533	2534	2535	2536	10,800
11,000	2537	2540	2541	2542	2543	2544	2545	2546	2547	2550	2551	2552	2553	2554	2555	2556	2557	2560	2561	2562	2563	2564	2565	2566	2567	11,000
11,200	2570	2571	2572	2573	2574	2575	2576	2577	2600	2601	2602	2603	2604	2605	2606	2607	2610	2611	2612	2613	2614	2615	2616	2617	2620	11,200
11,400	2621	2622	2623	2624	2625	2626	2627	2630	2631	2632	2633	2634	2635	2636	2637	2640	2641	2642	2643	2644	2645	2646	2647	2650	2651	11,400
11,600	2652	2653	2654	2655	2656	2657	2660	2661	2662	2663	2664	2665	2666	2667	2670	2671	2672	2673	2674	2675	2676	2677	2700	2701	2702	11,600
11,800	2703	2704	2705	2706	2707	2710	2711	2712	2713	2714	2715	2716	2717	2720	2721	2722	2723	2724	2725	2726	2727	2730	2731	2732	2733	11,800
12,000	2734	2735	2736	2737	2740	2741	2742	2743	2744	2745	2746	2747	2750	2751	2752	2753	2754	2755	2756	2757	2760	2761	2762	2763	2764	12,000
12,200	2765	2766	2767	2770	2771	2772	2773	2774	2775	2776	2777	3000	3001	3002	3003	3004	3005	3006	3007	3010	3011	3012	3013	3014	3015	12,200
12,400	3016	3017	3020	3021	3022	3023	3024	3025	3026	3027	3030	3031	3032	3033	3034	3035	3036	3037	3040	3041	3042	3043	3044	3045	3046	12,400
12,600	3047	3050	3051	3052	3053	3054	3055	3056	3057	3060	3061	3062	3063	3064	3065	3066	3067	3070	3071	3072	3073	3074	3075	3076	3077	12,600
12,800	3100	3101	3102	3103	3104	3105	3106	3107	3110	3111	3112	3113	3114	3115	3116	3117	3120	3121	3122	3123	3124	3125	3126	3127	3130	12,800
13,000	3131	3132	3133	3134	3135	3136	3137	3140	3141	3142	3143	3144	3145	3146	3147	3150	3151	3152	3153	3154	3155	3156	3157	3160	3161	13,000
13,200	3162	3163	3164	3165	3166	3167	3170	3171	3172	3173	3174	3175	3176	3177	3200	3201	3202	3203	3204	3205	3206	3207	3210	3211	3212	13,200
13,400	3213	3214	3215	3216	3217	3220	3221	3222	3223	3224	3225	3226	3227	3230	3231	3232	3233	3234	3235	3236	3237	3240	3241	3242	3243	13,400
13,600	3244	3245	3246	3247	3250	3251	3252	3253	3254	3255	3256	3257	3260	3261	3262	3263	3264	3265	3266	3267	3270	3271	3272	3273	3274	13,600
13,800	3275	3276	3277	3300	3301	3302	3303	3304	3305	3306	3307	3310	3311	3312	3313	3314	3315	3316	3317	3320	3321	3322	3323	3324	3325	13,800
14,000	3326	3327	3330	3331	3332	3333	3334	3335	3336	3337	3340	3341	3342	3343	3344	3345	3346	3347	3350	3351	3352	3353	3354	3355	3356	14,000
14,200	3357	3360	3361	3362	3363	3364	3365	3366	3367	3370	3371	3372	3373	3374	3375	3376	3377	3400	3401	3402	3403	3404	3405	3406	3407	14,200
14,400	3410	3411	3412	3413	3414	3415	3416	3417	3420	3421	3422	3423	3424	3425	3426	3427	3430	3431	3432	3433	3434	3435	3436	3437	3440	14,400
14,600	3441	3442	3443	3444	3445	3446	3447	3450	3451	3452	3453	3454	3455	3456	3457	3460	3461	3462	3463	3464	3465	3466	3467	3470	3471	14,600
14,800	3472	3473	3474	3475	3476	3477	3500	3501	3502	3503	3504	3505	3506	3507	3510	3511	3512	3513	3514	3515	3516	3517	3520	3521	3522	14,800
15,000	3523	3524	3525	3526	3527	3530	3531	3532	3533	3534	3535	3536	3537	3540	3541	3542	3543	3544	3545	3546	3547	3550	3551	3552	3553	15,000
15,200	3554	3555	3556	3557	3560	3561	3562	3563	3564	3565	3566	3567	3570	3571	3572	3573	3574	3575	3576	3577	3600	3601	3602	3603	3604	15,200
15,400	3605	3606	3607	3610	3611	3612	3613	3614	3615	3616	3617	3620	3621	3622	3623	3624	3625	3626	3627	3630	3631	3632	3633	3634	3635	15,400
15,600	3636	3637	3640	3641	3642	3643	3644	3645	3646	3647	3650	3651	3652	3653	3654	3655	3656	3657	3660	3661	3662	3663	3664	3665	3666	15,600
15,800	3667	3670	3671	3672	3673	3674	3675	3676	3677	3700	3701	3702	3703	3704	3705	3706	3707	3710	3711	3712	3713	3714	3715	3716	3717	15,800
16,000	3720	3721	3722	3723	3724	3725	3726	3727	3730	3731	3732	3733	3734	3735	3736	3737	3740	3741	3742	3743	3744	3745	3746	3747	3750	16,000
16,200	3751	3752	3753	3754	3755	3756	3757	3760	3761	3762	3763	3764	3765	3766	3767	3770	3771	3772	3773	3774	3775	3776	3777	4000	4001	16,200
16,400	4002	4003	4004	4005	4006	4007	4010	4011	4012	4013	4014	4015	4016	4017	4020	4021	4022	4023	4024	4025	4026	4027	4030	4031	4032	16,400

HIGH-ORDER OCTAL DIGITS

OCTAL-DECIMAL CONVERSION PROCEDURE

Consider the decimal number to be converted as a base and an increment. Locate the base (the next lower number which is evenly divisible by 200) in the margin of the lower chart and the increment in the body of the upper chart. The intersection of the row and column thus defined contains the high-order digits of the octal equivalent. The low-order digit appears in the margins of the upper chart opposite the increment. For example, to convert 7958 to octal, the base is 7800 and the increment is 158. Locate 158 in the upper chart and read down this column to the 7800 row below. The high-order octal result is 1742. Then read out to the margin of the upper chart to obtain the low-order digit of 6. Append (do not add) this digit to 1742 for an octal equivalent of 17,426.

To convert an octal number to decimal, locate the high-order digits in the body of the lower chart and the low-order digit in the margin of the upper chart. Then perform the converse of the above operation.

APPENDIX B

MISCELLANEOUS TABLES

Table B-1. Control Register Designations

CONTROL REGISTER	VARIANT CHARACTER (LCR & SCR Instructions)
CLC7	00
CLC1	01
CLC2	02
CLC3	03
CLC7'	04
CLC1'	05
CLC2'	06
CLC3'	07
SLC7	10
SLC1	11
SLC2	12
SLC3	13
SLC7'	14
SLC1'	15
SLC2'	16
SLC3'	17
CLC8	20
CLC4	21
CLC5	22
CLC6	23
CLC8'	24
CLC4'	25
CLC5'	26
CLC6'	27
SLC8	30
SLC4	31
SLC5	32
SLC6	33
SLC8'	34

Table B-1 (cont). Control Register Designations

CONTROL REGISTER	VARIANT CHARACTER (LCR & SCR Instructions)
SLC4'	35
SLC5'	36
SLC6'	37
AC0	--
AC1	--
AC2	--
AC3	--
CSR	64
EIR	66
AAR	67
BAR	70
IIR	76
SR	77

Table B-2. Extended Move (EXM) Conditions

CONDITIONS	VARIANT BITS					
	V_6	V_5	V_4	V_3	V_2	V_1
Type of Move						
1. A-field data bits → B	X	X	X	X	X	1
2. A-field word-mark bits → B	X	X	X	X	1	X
3. A-field item-mark bits → B	X	X	X	1	X	X
Direction of Move						
1. right to left	X	X	0	X	X	X
2. left to right	X	X	1	X	X	X
Termination of Move						
1. automatic after single-character move	0	0	X	X	X	X
2. A-field word mark	0	1	X	X	X	X
3. A-field item mark	1	0	X	X	X	X
4. A-field record mark	1	1	X	X	X	X

Table B-3. Branch on Condition Test (BCT) SENSE Switch Conditions

VARIANT CHARACTER (Octal)	BRANCH CONDITION
00	Unconditional
01	SENSE Switch 1 On
02	SENSE Switch 2 On
03	SENSE Switches 1 and 2 On
04	SENSE Switch 3 On
05	SENSE Switches 1 and 3 On
06	SENSE Switches 2 and 3 On
07	SENSE Switches 1, 2, and 3 On
10	SENSE Switch 4 On
11	SENSE Switches 1 and 4 On
12	SENSE Switches 2 and 4 On
13	SENSE Switches 1, 2, and 4 On
14	SENSE Switches 3 and 4 On
15	SENSE Switches 1, 3, and 4 On
16	SENSE Switches 2, 3, and 4 On
17	SENSE Switches 1, 2, 3, and 4 On
20	Unconditional
21	SENSE Switch 5 On
22	SENSE Switch 6 On
23	SENSE Switches 5 and 6 On
24	SENSE Switch 7 On
25	SENSE Switches 5 and 7 On
26	SENSE Switches 6 and 7 On
27	SENSE Switches 5, 6, and 7 On
30	SENSE Switch 8 On
31	SENSE Switches 5 and 8 On
32	SENSE Switches 6 and 8 On
33	SENSE Switches 5, 6, and 8 On
34	SENSE Switches 7 and 8 On
35	SENSE Switches 5, 7, and 8 On
36	SENSE Switches 6, 7, and 8 On
37	SENSE Switches 5, 6, 7, and 8 On

NOTE: When testing for a multiple SENSE switch condition, a branch occurs only if all of the specified conditions are met.

Table B-4. Branch on Condition Test (BCT) Indicator Conditions

VARIANT CHARACTER (Octal)	BRANCH CONDITION
40	Do not branch
41	$B < A$ (Low Compare)
42	$B=A$ (Equal Compare)
43	$B \leq A$ (Low or Equal Compare)
44	$B > A$ (High Compare)
45	$B \neq A$ (Unequal Compare)
46	$B \geq A$ (High or Equal Compare)
47	Unconditional
50	Overflow
51	Overflow or $B < A$
52	Overflow or $B=A$
53	Overflow or $B \leq A$
54	Overflow or $B > A$
55	Overflow or $B \neq A$
56	Overflow or $B \geq A$
57	Unconditional
60	Zero Balance
61	Zero Balance or $B < A$
62	Zero Balance or $B=A$
63	Zero Balance or $B \leq A$
64	Zero Balance or $B > A$
65	Zero Balance or $B \neq A$
66	Zero Balance or $B \geq A$
67	Unconditional
70	Overflow or Zero Balance
71	Overflow or Zero Balance or $B < A$
72	Overflow or Zero Balance or $B=A$
73	Overflow or Zero Balance or $B \leq A$
74	Overflow or Zero Balance or $B > A$
75	Overflow or Zero Balance or $B \neq A$
76	Overflow or Zero Balance or $B \geq A$
77	Unconditional

NOTE: When testing for a multiple indicator condition, a branch occurs if any one of the specified conditions is met.

Table B-5. Branch on Character Condition (BCC) Conditions

VARIANT CHARACTER (Octal)	BRANCH CONDITION
00	Unconditional
01*	A bit is 1
02	B bit is 1
03*	B and A bits are 11
04*	B and A bits are 00
05*	B and A bits are 01 (Positive sign)
06	B and A bits are 10 (Negative sign)
07*	B and A bits are 11 (same as 03)
10	Word-mark bit is 1
11*	Word-mark bit is 1, A bit is 1
12	Word-mark bit is 1, B bit is 1
13*	Word-mark bit is 1, B and A bits are 11
14*	Word-mark bit is 1, B and A bits are 00
15*	Word-mark bit is 1, Positive sign
16	Word-mark bit is 1, Negative sign
17*	Word-mark bit is 1, B and A bits are 11
20	Item-mark bit is 1
21*	Item-mark bit is 1, A bit is 1
22	Item-mark bit is 1, B bit is 1
23*	Item-mark bit is 1, B and A bits are 11
24*	Item-mark bit is 1, B and A bits are 00
25*	Item-mark bit is 1, Positive Sign
26	Item-mark bit is 1, Negative Sign
27*	Item-mark bit is 1, B and A bits are 11
30	Record mark
31*	Record mark, A bit is 1
32	Record mark, B bit is 1
33*	Record mark, B and A bits are 11
34*	Record mark, B and A bits are 00
35*	Record mark, Positive sign
36	Record mark, Negative sign
37*	Record mark, B and A bits are 11
40*	No punctuation (Word-mark and Item-mark bits are 00)
41*	No punctuation, A bit is 1
42*	No punctuation, B bit is 1
43*	No punctuation, B and A bits are 11
44*	No punctuation, B and A bits are 00
45*	No punctuation, Positive sign
46*	No punctuation, Negative sign
47*	No punctuation, B and A bits are 11
50*	Word mark only
51*	Word mark only, A bit is 1
52*	Word mark only, B bit is 1
53*	Word mark only, B and A bits are 11
54*	Word mark only, B and A bits are 00
55*	Word mark only, Positive sign
56*	Word mark only, Negative sign
57*	Word mark only, B and A bits are 11

Table B-5 (cont). Branch on Character Condition (BCC) Conditions

VARIANT CHARACTER (Octal)	BRANCH CONDITION
60*	Item mark only
61*	Item mark only, A bit is 1
62*	Item mark only, B bit is 1
63*	Item mark only, B and A bits are 11
64*	Item mark only, B and A bits are 00
65*	Item mark only, Positive sign
66*	Item mark only, Negative sign
67*	Item mark only, B and A bits are 11
70*t	Unconditional
71*t	Word mark or A bit is 1
72*t	Word mark or B bit is 1
73*t	Word mark or B and A bits are 11
74*t	Word mark or B and A bits are 00
75*t	Word mark or Positive sign
76*t	Word mark or Negative sign
77*t	Word mark or B and A bits are 11

*Valid only on systems equipped with the Advanced Programming Feature (Feature 010 or 011).

tThe Type 201 and 201-1 processors interpret variants 70 through 77 as if they were variants 30 through 37.

Table B-6. Series 200 Character Codes

Key Punch	Card Code	Central Processor Code	Octal	High Speed Printer	Key Punch	Card Code	Central Processor Code	Octal	High Speed Printer
0	0	000000	00	0	$\bar{0}$ or -	X, 0 or X(1)	100000	40	-
1	1	000001	01	1	J	X, 1	100001	41	J
2	2	000010	02	2	K	X, 2	100010	42	K
3	3	000011	03	3	L	X, 3	100011	43	L
4	4	000100	04	4	M	X, 4	100100	44	M
5	5	000101	05	5	N	X, 5	100101	45	N
6	6	000110	06	6	O	X, 6	100110	46	O
7	7	000111	07	7	P	X, 7	100111	47	P
8	8	001000	10	8	Q	X, 8	101000	50	Q
9	9	001001	11	9	R	X, 9	101001	51	R
	8, 2	001010	12	'		X, 8, 2	101010	52	#
#	8, 3	001011	13	=	$	X, 8, 3	101011	53	$
@	8, 4	001100	14	:	*	X, 8, 4	101100	54	*
Space	Blank	001101	15	Blank		X, 8, 5	101101	55	"
	8, 6	001110	16	> (2)		X, 8, 6	101110	56	≠ (2)
	8, 7	001111	17	&	- or $\bar{0}$	X or X, 0(1)	101111	57	1/2 or ! (2) (3)
$\overset{\&}{0}$ or &	R, 0 or R(1)	010000	20	+		8, 5	110000	60	< (2)
A	R, 1	010001	21	A	/	0, 1	110001	61	/
B	R, 2	010010	22	B	S	0, 2	110010	62	S
C	R, 3	010011	23	C	T	0, 3	110011	63	T
D	R, 4	010100	24	D	U	0, 4	110100	64	U
E	R, 5	010101	25	E	V	0, 5	110101	65	V
F	R, 6	010110	26	F	W	0, 6	110110	66	W
G	R, 7	010111	27	G	X	0, 7	110111	67	X
H	R, 8	011000	30	H	Y	0, 8	111000	70	Y
I	R, 9	011001	31	I	Z	0, 9	111001	71	Z
	R, 8, 2	011010	32	;		0, 8, 2	111010	72	@
.	R, 8, 3	011011	33	.	,	0, 8, 3	111011	73	,
⌑	R, 8, 4	011100	34	)	%	0, 8, 4	111100	74	(
	R, 8, 5	011101	35	%		0, 8, 5	111101	75	C_R
	R, 8, 6	011110	36	■		0, 8, 6	111110	76	⌑ (2)
& or $\overset{\&}{0}$	R or R, 0(1)	011111	37	? (2)		0, 8, 7	111111	77	¢ (2)

(1) Special Code (for use with H-400/1400 and H-800/1800 cards). The second (alternative) card code is equivalent to the stated central processor code when control character 26 is coded in a card read or punch PCB instruction.

(2) Indicates symbol which will be printed by a printer which has a 63-character drum (Type 222 printers).

(3) The exclamation point replaces the one-half symbol on a type roll containing the Mark II character font.

Table B-7. Binary, Octal, and Decimal Equivalents

BIN.	OCT.	DEC.	BIN.	OCT.	DEC.
0	0	0	100000	40	32
1	1	1	100001	41	33
10	2	2	100010	42	34
11	3	3	100011	43	35
100	4	4	100100	44	36
101	5	5	100101	45	37
110	6	6	100110	46	38
111	7	7	100111	47	39
1000	10	8	101000	50	40
1001	11	9	101001	51	41
1010	12	10	101010	52	42
1011	13	11	101011	53	43
1100	14	12	101100	54	44
1101	15	13	101101	55	45
1110	16	14	101110	56	46
1111	17	15	101111	57	47
10000	20	16	110000	60	48
10001	21	17	110001	61	49
10010	22	18	110010	62	50
10011	23	19	110011	63	51
10100	24	20	110100	64	52
10101	25	21	110101	65	53
10110	26	22	110110	66	54
10111	27	23	110111	67	55
11000	30	24	111000	70	56
11001	31	25	111001	71	57
11010	32	26	111010	72	58
11011	33	27	111011	73	59
11100	34	28	111100	74	60
11101	35	29	111101	75	61
11110	36	30	111110	76	62
11111	37	31	111111	77	63

Table B-8. Powers of 2

n	2^n
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1 024
11	2 048
12	4 096
13	8 192
14	16 384
15	32 768
16	65 536
17	131 072
18	262 144
19	524 288
20	1 048 576
21	2 097 152
22	4 194 304
23	8 388 608
24	16 777 216

Table B-9. Move or Scan Variants

MOVE OPERATION CODES

Mnemonic Op Code	Statement Name	Corresponding Move or Scan Variant
MLC	Move Left Characters	63
MLN	Move Left Numerics	61
MLW	Move Left Word Marks	64
MLZ	Move Left Zones	62
MLCA	Move Left Characters to A-Field Word Mark	23
MLCB	Move Left Characters to B-Field Word Mark	43
MLCS	Move Left Character Single	03
MLCW	Move Left Characters and Word Marks	67
MLNA	Move Left Numerics to A-Field Word Mark	21
MLNB	Move Left Numerics to B-Field Word Mark	41
MLNS	Move Left Numeric Single	01
MLNW	Move Left Numerics and Word Marks	65
MLWA	Move Left Word Marks to A-Field Word Mark	24
MLWB	Move Left Word Mark to B-Field Word Mark	44
MLWS	Move Left Word Mark Single	04
MLZA	Move Left Zones to A-Field Word Mark	22
MLZB	Move Left Zones to B-Field Word Mark	42
MLZS	Move Left Zone Single	02
MLZW	Move Left Zones and Word Marks	66
MLCWA	Move Left Characters and Word Mark to A-Field Word Mark	27
MLCWB	Move Left Characters and Word Mark to B-Field Word Mark	47
MLCWS	Move Left Characters and Word Mark Single	07
MLNWA	Move Left Numerics and Word Mark to A-Field Word Mark	25
MLNWB	Move Left Numerics and Word Mark to B-Field Word Mark	45
MLNWS	Move Left Numeric and Word Mark Single	05
MLZWA	Move Left Zones and Word Mark to A-Field Word Mark	26
MLZWB	Move Left Zones and Word Mark to B-Field Word Mark	46
MLZWS	Move Left Zones and Word Mark Single	06
MRC	Move Right Characters	13

Table B-9 (cont). Move or Scan Variants

MOVE OPERATION CODES

Mnemonic Op Code	Statement Name	Corresponding Move or Scan Variants
MRN	Move Right Numerics	11
MRW	Move Right Word Marks	14
MRZ	Move Right Zones	12
MRCG	Move Right Characters to A-Field Group Mark-Word Mark	53
MRCM	Move Right Characters to A-Field Record Mark or Group Mark-Word Mark	73
MRCR	Move Right Characters to A-Field Record Mark	33
MRCW	Move Right Characters and Word Mark to A- or B-Field Word Mark	17
MRNG	Move Right Numerics to A-Field Group Mark-Word Mark	51
MRNM	Move Right Numerics to A-Field Record Mark or Group Mark-Word Mark	71
MRNR	Move Right Numerics to A-Field Record Mark	31
MRNW	Move Right Numerics and Word Mark to A- or B-Field Word Mark	15
MRWG	Move Right Word Marks to A-Field Group Mark-Word Mark	54
MRWM	Move Right Word Marks to A-Field Record Mark or Group Mark-Word Mark	74
MRWR	Move Right Word Marks to A-Field Record Mark	34
MRZG	Move Right Zones to A-Field Group Mark-Word Mark	52
MRZM	Move Right Zones to A-Field Record Mark or Group Mark-Word Mark	72
MRZR	Move Right Zones to A-Field Record Mark	32
MRZW	Move Right Zones and Word Mark to A- or B-Field Word Mark	16
MRCWG	Move Right Characters and Word Marks to A-Field Group Mark-Word Mark	57
MRCWM	Move Right Characters and Word Marks to A-Field Record Mark-Group Mark-Word Mark	77
MRCWR	Move Right Characters and Word Marks to A-Field Record Mark	37
MRNWG	Move Right Numerics and Word Marks to A-Field Group Mark-Word Mark	55
MRNWM	Move Right Numerics and Word Marks to A-Field Record Mark-Group Mark-Word Mark	75

Table B-9 (cont). Move or Scan Variants

MOVE OPERATION CODES

Mnemonic Op Code	Statement Name	Corresponding Move or Scan Variants
MRNWR	Move Right Numerics and Word Marks to A-Field Record Mark	35
MRZWG	Move Right Zones and Word Marks to A-Field Group Mark-Word Mark	56
MRZWM	Move Right Zones and Word Marks to A-Field Record Mark-Group Mark-Word Mark	76
MRZWR	Move Right Zones and Word Marks to A-Field Record Mark	36
	SCAN OPERATION CODES	
SCNL	Scan Left to A- or B-Field Word Mark	60
SCNR	Scan Right to A- or B-Field Word Mark	10
SCNLA	Scan Left to A-Field Word Mark	20
SCNLB	Scan Left to B-Field Word Mark	40
SCNLS	Scan Left Single Position	00
SCNRG	Scan Right to A-Field Group Mark-Word Mark	50
SCNRM	Scan Right to A-Field Record Mark or Group Mark-Word Mark	70
SCNRR	Scan Right to A-Field Record Mark	30

APPENDIX C

INSTRUCTION SUMMARY

INSTRUCTIONS FORMATS AND TIMING

Each Series 200 instruction is described in terms of its operation code, formats, and timing formulas for the Series 200 Models 200/1200/1250/2200 in Table C-1. In addition, reference is made in each case to the page where the operations initiated by the instruction are described.

Preliminary timing formulas for the Model 4200 are given in Table C-2. Since the internal operation of the Model 4200 processor differs from that of the other Series 200 processors in that data is moved in groups of four characters (a word) rather than singly, the 4200 timing formulas differ considerably from those of the other processors.

The formulas given in both tables provide execution time in memory cycles. Equivalent expressions for symbols used in the tables are as follows:

SYMBOL	MEANING
A	Address of A-operand field.
B	Address of B-operand field.
h	The sum of the values of the multiplier digits which are less than or equal to five, plus the sum of the elevens complements of all digits whose values are greater than five.
N_a	Number of characters in the A-operand field.
N_{aw}	Number of words in the A-operand field.
N_b	Number of characters in the B-operand field.
N_{bw}	Number of words in the B-operand field.
N_{b1}	Number of words that the A field occupies in the B field, whether or not the operands have been modified by an arithmetic operation.
N_{b2}	Number of words in the B-operand field excluding N_{b1}.
N_c	Number of control characters in the instruction.
N_{cn}	Number of control characters following control character 3 (C3).
N_{dd}	Number of digits in the dividend.
N_i	Number of characters in the instruction.

SYMBOL	MEANING
N_{ia}	Number of words in the item to be translated.
N_{ib}	Number of words in the result item.
N_{ic}	Number of translation units (6-bit or 12-bit characters) to be translated.
N_j	Number of character locations bypassed to reach the next sequential op code.
N_m	Number of characters moved.
N_{mr}	Number of digits in the multiplier.
N_q	Number of digits in the quotient $(=N_{dd}-Z_{ld}-N_a+Z_{la}+1)$.
N_r	Number of characters referenced.
N_{sc}	Number of characters scanned.
N_{st}	Number of characters stored.
N_w	Number of characters in the A- or B-operand field, whichever is shorter.
N_{wj}	Number of words bypassed to reach the next sequential op code.
N_{ws}	Number of words stored.
n	Number of items in the table or the number of times the A operand is compared against some portion of the B operand.
Q_i	The value of the "i^{th}" digit of the quotient.
s	Sum of all multiplier digits.
SUM	Sum of the upwards-rounded values of all multiplier digits divided by 2.
V	Variant character.
W	Number of memory words used to store the data involved.
W_i	Number of four-character words used to store one more than the total number of characters in the instruction.
W_{mr}	Number of words in the multiplier.
X_0	Zero if no second scan (zero suppression); one if the scan is performed.
Y_0	Zero if no third scan (dollar-sign insertion); one if the scan is performed.
Z	Number of characters scanned during zero suppression.
Z_{la}	Number of leading zeros in the A-operand field.
Z_{law}	Number of words containing leading zeros in the A-operand field.
Z_{ld}	Number of leading zeros in the dividend

SYMBOL	MEANING
Z_{mr}	Number of zeros in the multiplier.
Z_{ta}	Number of trailing zeros (i. e., consecutive low-order zeros) in the A-operand field.
Z_{taw}	Number of words containing trailing zeros in the A-operand field.
Z_w	Number of words scanned during zero suppression.
Z_z	Zero if $Z_{1a} = 0$; one if $Z_{1a} \neq 0$.
$\$$	Number of characters scanned during dollar-sign insertion.
$\$_w$	Number of words scanned during dollar-sign insertion.

NOTE: The timing formulas presented in Tables C-1, C-2, and C-3 are based on the use of direct addressing. If address modification is used, the formulas in Tables C-1 and C-3 for the Models 200, 1200, 1250, and 2200 should be modified as follows:

1. Indirect Addressing — Add one memory cycle for each character extracted as a result of indirect addressing.
2. Indexed Addressing — Add three memory cycles for each indexed address.

Likewise, the use of address modification requires that the formulas in Tables C-2 and C-3 for the Model 4200 be modified as follows:

1. Indirect Addressing — Add 1.16 memory cycles for each indirect address formed plus one memory cycle for each word extracted as a result of indirect addressing.
2. Indexed Addressing — Add 3.167 memory cycles if one address is indexed, 5.16 memory cycles if both addresses are indexed.

Table C-1. Instruction Summary — Timing Formulas for Models 200, 1200, 1250, 2200*

Op Code				Function	Timing (Memory Cycles)[1]	Format	Extraction Path[2]	Required Word Marks	Terminated By	Can Instruction Be Chained?	Described On Pages
Mnemonic	Octal	Card Code	Key Punch								
					ARITHMETIC INSTRUCTIONS						
A	36	R, 8, 6		Decimal Add	$N_i+2+N_w+2N_b$ (no recomplement)[4] $N_i+2+N_w+4N_b$ (recomplement)[4]	a. A/A, B b. A/A c. A/	Duplicates A.	B operand. A operand only if smaller than B.	B-operand word mark.	Yes.	8-14
S	37	R or R, 0[3]	&	Decimal Subtract	$N_i+2+N_w+2N_b$ (no recomplement)[4] $N_i+2+N_w+4N_b$ (recomplement)[4]	a. S/A, B b. S/A c. S/	Duplicates A.	B operand. A operand only if smaller than B.	B-operand word mark.	Yes.	8-16
BA	34	R, 8, 4	□	Binary Add	$N_i+1+N_w+2N_b$	a. BA/A, B b. BA/A c. BA/	Duplicates A.	B operand. A operand only if smaller than B.	B-operand word mark.	Yes.	8-17
BS	35	R, 8, 5		Binary Subtract	$N_i+1+N_w+2N_b$	a. BS/A, B b. BS/A c. BS	Duplicates A.	B operand. A operand only if smaller than B.	B-operand word mark.	Yes.	8-19
ZA	16	8, 6		Zero and Add	$N_i+1+N_w+N_b$	a. ZA/A, B b. ZA/A c. ZA/	Duplicates A.	B operand. A operand only if smaller than B.	B-operand word mark.	Yes.	8-20
ZS	17	8, 7		Zero and Subtract	$N_i+1+N_w+N_b$	a. ZS/A, B b. ZS/A c. ZS/	Duplicates A.	B operand. A operand only if smaller than B.	B-operand word mark.	Yes.	8-22
M	26	R, 6	F	Decimal Multiply	See Table C-3.	a. M/A, B b. M/A c. M/	Preserves B.	A and B fields.	Both word marks.	Yes.	8-23
D	27	R, 7	G	Decimal Divide	See Table C-3.	a. D/A, B b. D/A c. D/	Preserves B.	A operand (divisor).	A-operand word mark.	Yes.	8-25
					LOGIC INSTRUCTIONS						
EXT	31	R, 9	I	Extract (Logical Product)	N_i+1+3N_w	a. EXT/A, B b. EXT/A c. EXT/	Preserves B.	Smaller operand.	Word mark of smaller operand.	Yes.	8-28
HA	30	R, 8	H	Half Add (Exclusive Or)	N_i+1+3N_w	a. HA/A, B b. HA/A c. HA/	Preserves B.	Smaller operand.	Word mark of smaller operand.	Yes.	8-29
SST	32	R, 8, 2		Substitute	N_i+4	a. SST/A, B, V b. SST/A, B[5] c. SST/A d. SST/	Preserves B.	None.	Single-character operation.	Yes.	8-30
C	33	R, 8, 3		Compare	$N_i+2+N_w+N_b$[6]	a. C/A, B b. C/A c. C/	Preserves B.	B operand. A operand only if smaller than B.	B-operand word mark.	Yes.	8-32
B	65	0, 5	V	Branch (Unconditional)	N_i+2[6]	a. B/A	Bypasses B.	None.	n/a	No.	8-34
BCT	65	0, 5	V	Branch on Condition Test	N_i+2[6]	a. BCT/A, V[7] b. BCT/	Bypasses B.	None.	n/a	Yes.[8]	8-35
BCC	54	X, 8, 4	*	Branch on Character Condition	N_i+4	a. BCC/A, B, V[5] b. BCC/A, B c. BCC/A	Preserves B.	None.	Single-character operation.	Yes.	8-39
BCE	55	X, 8, 5		Branch if Character Equal	N_i+4	a. BCE/A, B, V[5] b. BCE/A, B c. BCE/A d. BCE/	Preserves B.	None.	Single-character operation.	Yes.[9]	8-42
BBE	56	X, 8, 6		Branch on Bit Equal	N_i+4	a. BBE/A, B, V b. BBE/A, B c. BBE/A d. BBE/	Preserves B.	None.	Single-character operation.	Yes.	8-44
					CONTROL INSTRUCTIONS						
SW	22	R, 2	B	Set Word Mark	N_i+3[10]	a. SW/A, B b. SW/A c. SW/	Duplicates A.	None.	n/a	Yes.	8-48
SI	20	R, 0 or R[3]	&	Set Item Mark	N_i+3[10]	a. SI/A, B b. SI/A c. SI/	Duplicates A.	None.	n/a	Yes.	8-49
CW	23	R, 3	C	Clear Word Mark	N_i+3	a. CW/A, B b. CW/A c. CW/	Duplicates A.	Word marks are cleared.	n/a	Yes.	8-50
CI	21	R, 1	A	Clear Item Mark	N_i+3	a. CI/A, B b. CI/A c. CI/	Duplicates A.	None.	n/a	Yes.	8-51
H	45	X, 5	N	Halt	N_i+2[6]	a. H/[11] b. H/A c. H/A, B d. H/A, B, V	Preserves B.	None.	n/a	No.	8-52
NOP	40		-	No Operation	N_i+2[12]	a. NOP/	Bypasses A and B.	None.	n/a	No.	8-54
MCW	14	8, 4		Move Characters to Word Mark	N_i+1+2N_w	a. MCW/A, B b. MCW/A c. MCW	Preserves B.	Smaller operand.	Word mark of smaller operand.	Yes.	8-55

Table C-1 (cont). Instruction Summary — Timing Formulas for Models 200, 1200, 1250, 2200*

Op Code				Function	Timing (Memory Cycles)[1]	Format	Extraction Path[2]	Required Word Marks	Terminated By:	Can Instruction Be Chained?	Described On Page:
Mnemonic	Octal	Card Code	Key Punch								
CONTROL INSTRUCTIONS (cont)											
LCA	15	Blank	Space	Load Characters to A-Field Word Mark	N_i+1+2N_a	a. LCA/A, B b. LCA/A c. LCA/	Preserves B.	A operand.	A-operand word mark.	Yes.	8-56
SCR	24	R, 4	D	Store Control Registers	N_i+5[6]	a. SCR/A, V[7] b. SCR/A c. SCR/	Bypasses B.	None.	n/a	Yes.[8]	8-58
LCR	25	R, 5	E	Load Control Registers	N_i+5[6]	a. LCR/A, V[7] b. LCR/A c. LCR/	Bypasses B.	None.	n/a	Yes.[8]	8-60
CAM	42	X, 2	K	Change Addressing Mode	N_i+2[12]	a. CAM/V[7] b. CAM/	Bypasses A and B.	None.	n/a	Yes.[8]	8-62
CSM	43	X, 3	L	Change Sequencing Mode	N_i+3[12]	a. CSM/[11] b. CSM/A c. CSM/A, B d. CSM/A, B, V	Preserves B.	None.	n/a	Yes.[8]	8-66
EXM	10	8	8	Extended Move	N_i+1+2N_a	a. EXM/A, B, V[5] b. EXM/A, B c. EXM/A d. EXM/	Preserves B.	See page 8-67	See page 8-67	Yes.[8]	8-67
MAT	60	8, 5		Move and Translate	N_i+3N_a[13]	a. MAT/A,B,V_1,V_2 b. MAT/A, B, C	See page 8-70	A operand.	Word mark in A operand or in table.	No.	8-70
MIT	62	0, 2	S	Move Item and Translate	$N_i+N_a+2N_{ic}$[13]	a. MIT/A, B, V_1, V_2, V_3 b. MIT/A,B, C, V_1	See page 8-74	None.	A-operand item mark or word mark in table.	No.	8-74
LIB	77	0, 8, 7		Load Index/ Barricade Register	N_i+3 N_i+5	a. LIB/A b. LIB/A/B	Preserves B.	None.	Single-character operation.	Yes.	8-79
SIB	76	0, 8, 6		Store Index/ Barricade Register	N_i+3 N_i+5	a. SIB/A b. SIB/A/B	Preserves B.	None.	Single-character operation.	Yes.	8-82
TLU	57	R, 6	F	Table Lookup	$N_i+1+n(N_a)+N_b$[14]	a. TLU/A, B, V b. TLU/A, B c. TLU/A d. TLU	Preserves B.	A operand.	A-operand word mark.	Yes.	8-83
MOS	13	8, 3	#	Move or Scan	$N_i+1+3(N_m)$ (14)(Move) $N_i+1+3(N_{sc})$ (1200 Scan) $N_i+2+2(N_{sc})$ (2200 Scan)	a. MOS/A, B, Y b. MOS/A, B c. MOS/A d. MOS	Preserves B	See page 8-86	See page 8-86	Yes.	8-86
INTERRUPT CONTROL INSTRUCTIONS											
SVI	46	X, 6	O	Store Variant and Indicators	$N_i+2+N_{st}+N_j$[15]	a. SVI/V	Bypasses A and B.	See page 8-93	See page 8-93	No.	8-92
RVI	67	0, 7	X	Restore Variant and Indicators	N_i+2+N_r[4]	a. RVI/A, V	Restores A and bypasses B.	None.	Word mark of next instruction.	No.	8-95
MC	44	X, 4	M	Monitor Call	N_i+2[4]	a. MC/	Bypasses A and B.	None.	Word mark of next instruction.	No.	8-98
RNM	41	X, 1	J	Resume Normal Mode	N_i+3[16]	a. RNM/A, B b. RNM/A c. RNM/	Preserves B.	None.	n/a	No.	8-99
EDITING INSTRUCTION											
MCE	74	0, 8, 4	%	Move Characters and Edit	$N_i+1+N_a+2N_b+2Z+2\$$	a. MCE/A, B[7] b. MCE/A c. MCE/	Preserves B.	A operand and B operand (see page 8-106	See page 8-106	No.	8-104
INPUT/OUTPUT INSTRUCTIONS											
PDT	66	0, 6	W	Peripheral Data Transfer	MODEL 200: N_i+1 + data transfer time. MODELS 1200 and 1250 $(N_i-N_c+1)+(N_{cn}+3)$ input/output cycles + 1 processor cycle + data transfer time.[17] MODEL 2200: $(N_i-N_c+1)+2N_c$ + data transfer time.	a. PDT/A,C_1,. C_n	Bypasses B.	None.	Record mark in memory or unit record length.	No.	8-115

Table C-1 (cont). Instruction Summary — Timing Formulas for Models 200 1200, 1250, 2200*

ToP

Op Code				Function	Timing (Memory Cycles)[1]	Format	Extraction Path[2]	Required Word Marks	Terminated By:	Can Instruction Be Chained?	Described On Page:
Mnemonic	Octal	Card Code	Key Punch								
					INPUT/OUTPUT INSTRUCTIONS (cont)						
PCB	64	0, 4	U	Peripheral Control and Branch	MODEL 200: N_i+1 (if no branch condition exists) N_i+2 (if a branch occurs) MODELS 1200 and 1250: $(N_i-N_c+1)+N_c$ input/output cycles[17] MODEL 2200: $(N_i-N_c+1)+2N_c$	a. PCB/A, $C_1 \ldots C_n$	Bypasses B.	None.	n/a	No.	8-127

*All information given in this table, other than timing formulas, is applicable to the Model 4200. Timing formulas for the 4200 are presented in Table C-2. See Table F-1 for information concerning Scientific Unit instructions.

[1] Except where otherwise indicated, add one memory cycle to each of these formulas if the instruction is being executed in a Type 2201 processor.

[2] The extraction path of the various instructions is defined as follows:

- Preserves B — The previous contents of BAR are used as the B address when the instruction is coded in the format Op Code/A.
- Duplicates A — The contents of AAR are used as the B address when the instruction is coded in the format Op Code/A.
- Bypasses B — The contents of BAR are not used in any format.
- Bypasses A and B — The contents of AAR and BAR are not used in any format.

[3] The second (alternate) card code is in effect when control character 26 is coded in a Card Read or Punch PCB instruction.

[4] Subtract one memory cycle from this formula if the instruction is executed in a Type 1201 or 1251 processor.

[5] This instruction can be coded only in formats a. and d. when issued in a Type 201 or 201-1 processor.

[6] Add two memory cycles to this formula if the instruction is executed in a Type 2201 processor.

[7] This instruction can be coded only in format a. when issued in a Type 201 or 201-1 processor.

[8] This instruction cannot be chained in the Type 201 or 201-1 processor.

[9] This instruction can be chained in the Type 201 or 201-1 processor only if the preceding instruction is also a BCE instruction.

[10] Subtract one memory cycle from this formula if the instruction is issued in the Type 1201 or 1251 processor in the format Op Code/A, B.

[11] This instruction can be coded only in formats a., b., and c. when issued in a Type 201 or 201-1 processor.

[12] Subtract one memory cycle from this formula if the instruction is executed in a Type 1201 or 1251 processor.

[13] If the instruction is executed in a Type 2201 processor, do not add the one memory cycle mentioned in footnote 1.

[14] This formula applies only to the Type 1201, 1251, and 2201 processors; this instruction is not available with the Model 200 processors.

[15] Subtract one memory cycle from this formula if the instruction is executed in a Model 200 processor.

[16] Add two memory cycles to this formula if the instruction is executed in a Type 2201 processor. Subtract one memory cycle from the formula if the instruction is executed in a Type 1201 or 1251 processor.

[17] The "processor cycle" is the one memory cycle out of every four which is given unconditionally to the processor for internal operations; the three remaining cycles are termed "input/output cycles."

Table C-2. Instruction Timings for the Model 4200

Mnemonic Op Code	Instruction Name	Timing Formulas	Notes
Fixed-Point Arithmetic Instructions			
A	Decimal Add	No Recomplement $W_i+N_{aw}+2N_{b1}+2.5N_{b2}+K^1$ Recomplement $W_i+N_{aw}+2.5N_{bw}+2N_{b1}+2.5N_{b2}+K^1$	BC K = 5 AV K = 6.5 WC K = 7
S	Decimal Subtract	No Recomplement $W_i+N_{aw}+2N_{b1}+2.5N_{b2}+K^1$ Recomplement $W_i+N_{aw}+2.5N_{bw}+2N_{b1}+2.5N_{b2}+K^1$	
BA	Binary Add	$W_i+N_{aw}+2N_{b1}+2.5N_{b2}+K^1$	BC K = 5 AV K = 6.5 WC K = 7.5
BS	Binary Subtract	$W_i+N_{aw}+2N_{b1}+2.5N_{b2}+K^1$	
ZA	Zero and Add	$W_i+2N_{b1}+N_{b2}+K^2$	N_{b1} value is even K = 6 N_{b1} value is odd K = 7 S Mode: Add 2 cycles
ZS	Zero and Subtract	$W_i+2N_{b1}+N_{b2}+K^2$	
M	Decimal Multiply	See Table C-3.	
D	Decimal Divide	See Table C-3.	
Logic Instructions			
EXT	Extract	$W_i+3N_{b1}+7$	
HA	Half Add	$W_i+3N_{b1}+7$	
SST	Substitute	$W_i+7.34$	
C	Compare	$W_i+2N_{b1}+N_{b2}+K^1$	BC K = 5.5 AV K = 7 WC K = 8
B	Branch (unconditional)	$W_i+4.5$	

Table C-2 (cont). Instruction Timings for the Model 4200

Mnemonic Op Code	Instruction Name	Timing Formulas	Notes
Logic Instructions (cont)			
BCT	Branch on Condition Test	$W_i+4.5$	
BCC	Branch on Character Condition	W_i+6	
BCE	Branch on Character Equal	W_i+6	
BBE	Branch on Bit Equal	W_i+6	
Control Instructions			
SW	Set Word Mark	W_i+5	
SI	Set Item Mark	W_i+5	
CW	Clear Word Mark	W_i+5	
CI	Clear Item Mark	W_i+5	
H	Halt	W_i+5	
NOP	No Operation	W_i+4	
MCW	Move Characters to Word Mark	$W_i+2N_{bl}+K^2$	N_{bl} value is even $K = 6$ N_{bl} value is odd $K = 7$
LCA	Load Characters to A-field Word Mark	$W_i+2N_{bl}+K^2$	N_{bl} value is even $K = 5$ N_{bl} value is odd $K = 6$
SCR	Store Control Registers	$W_i+W+4.33$ W_i+W+13	Non-I/O register I/O register
LCR	Load Control Registers	$W_i+W+4.33$ W_i+W+24	Non-I/O register I/O register
CAM	Change Addressing Mode	W_i+4	

Table C-2 (cont). Instruction Timings for the Model 4200

Mnemonic Op Code	Instruction Name	Timing Formulas	Notes
Control Instructions (cont)			
CSM	Change Sequencing Mode	W_i+4	
EXM	Extended Move	$W_i+2N_{bl}+K^2$	N_{bl} value is even K=5 N_{bl} value is odd K=6
MAT	Move and Translate	$W_i+1.67N_{ia}+1.67N_{ib}+N_{ic}+6.5$	If A < 9+B then N_{ia} = Number of characters in the item to be translated. N_{ib} = Number of characters in the result item.
MIT	Move Item and Translate	$W_i+1.67N_{ia}+1.67N_{ib}+N_{ic}+6.5$	If B < 9+A or if the translation is 6-bit to 12-bit (or 12-bit to 6-bit) then N_{ia} = Number of characters in the item to be translated. N_{ib} = Number of characters in the result item.
LIB	Load Index/Barricade Register	Basic storage protection $W_i+W+4.5$ Storage protection with base relocation W_i+2W+6	
SIB	Store Index/Barricade Register	Basic storage protection W_i+W+4 Storage protection with base relocation W_i+2W+5	
TLU	Table Lookup	$W_i+n(N_{aw})+N_{bw}+3n+9$	
MOS	Move or Scan	$W_i+3+(3N_m+1)$	

Table C-2 (cont). Instruction Timings for the Model 4200

Mnemonic Op Code	Instruction Name	Timing Formulas	Notes
	Interrupt Control Instructions		
MC	Monitor Call	W_i+4	
SVI	Store Variant and Indicators	$W_i+N_{ws}+N_{wj}+8$	
RVI	Restore Variant and Indicators	W_i+7	
RNM	Resume Normal Mode	$W_i+4.5$	
	Edit Instruction		
MCE	Move Characters and Edit	$W_i+N_{aw}+2.3N_{bw}+2Z_w+2\$_w+6+X_o+Y_o$	

Table C-3. Timings for Decimal Multiply and Divide, Models 200, 1200, 1250, 2200, and 4200

Function	Model	Timing (Memory Cycles)
Multiply	200 1200 and 1250	$N_i+5+2N_a+2Z_{ta}+5N_{mr}-Z_{mr}+s(N_a-Z_{ta})+2(N_a-Z_{ta})(N_{mr}-Z_{mr})$
	2200	$N_i+8+2N_a+2Z_{ta}+5N_{mr}-Z_{mr}+\text{sum}(N_a-Z_{ta})+3(N_a-Z_{ta})(N_{mr}-Z_{mr})$
	4200	$W_i+2N_{aw}+\left[N_{mr}-Z_{mr}\right]\left[1.1N_a-Z_{ta}+8.0\right]+\frac{1}{3}(S)(N_a-Z_{ta})$ $+2.3Z_{mr}+Z_{taw}+13$
Divide	200 1200 and 1250	N_i+4+2N_a if divisor = 0 $N_i+17.5+4.5N_a+15.5Z_{la}+12.5N_{dd}+15N_a(N_{dd}-N_a+Z_{la})$ if $(N_a-Z_{la}) \le (N_{dd})$ and divisor ≠ 0 N_i+7+4N_a if $(N_a-Z_{la}) > (N_{dd})$
	2200	N_i+7+2N_a if divisor = 0 $N_i+9+2Z_z+5N_a+3Z_{ld}+N_q+15N_a-2Z_{la}+18.25)$ if $(N_a-Z_{la}) \le (N_{dd}-Z_{ld})$ and divisor ≠ 0 $N_i+9+2N_a+2N_{dd}$ if $N_a > N_{dd}$ and $(N_a-Z_{la}) > (N_{dd}-Z_{ld})$ $N_i+10+N_a+3N_{dd}$ if $N_a \ge N_{dd}$ and $(N_a-Z_{la}) > (N_{dd}-Z_{ld})$
	4200	$W_i+2N_{aw}+Z_{law}\sum_{i=1}^{N_q}(Q_1+2)(3N_{aw}+2)+19$

APPENDIX D

INTERRUPT PROCESSING

The execution of main-program instructions by the processor can be interrupted by an external interrupt source and, if the processor is a Type 1201, 1251 or 2201 equipped with the Storage Protect Feature (see Appendix E) or a Type 4201 equipped with the Extended Multiprogramming and 8-Bit Transfer Feature (see Appendix G), by an internal interrupt source.

EXTERNAL INTERRUPT

An external interrupt signal can be generated by any or all of three sources:

1. The operator's control panel or console;
2. The Monitor Call instruction (see page 8-98); or
3. A peripheral control.

The first two sources interrupt the processor directly: in the case of the control panel or console, the operator simply presses the INTERRUPT button; the Monitor Call instruction interrupts the processor when it is executed. However, a peripheral control interrupts program sequence as directed by the settings of two programmable storage functions contained within the control, as described on page D-5.

The interrupt signal sets indicators to show the source (whether 1., 2., or 3., above) and the type (external) of interruption. These indicators can be stored and then tested by programmed instruction as described further in this appendix. The processor acts upon the interrupt signal when the following conditions are present:

1. The processor is in the RUN mode (i.e., the processor is executing, without manual intervention, stored-program instructions under control of SR).
2. The processor is not in the external interrupt mode.
3. An instruction op code is about to be extracted.
4. A memory cycle is allocated to the processor.

It should be noted that condition 3. above does not cause an extensive delay if a Type 201-2, 1201, 2201, or 4201 processor is attempting to extract a Peripheral Data Transfer (PDT) instruction and the specified read/write channel or peripheral control is "busy." The attempt to issue a PDT instruction to a busy read/write channel or peripheral control does not "stall" the central processor. Rather, the instruction is "re-extracted": SR is set back to the address of the PDT op code, so that condition 3. recurs immediately after the channel or control is found busy.

When the central processor is interrupted, it performs the following functions:

1. Stores the current status of the arithmetic, comparison, address mode, and trap mode indicators in the auxiliary indicators register (AIR).
2. Clears the arithmetic indicators.
3. Enters the three-character, non-trap mode.
4. Interchanges the contents of SR and EIR and branches to the instruction whose op code address was previously stored in EIR.
5. Enters the external interrupt mode.

The interrupt signal is maintained until one of the following steps is taken:

1. A PDT instruction is issued to the peripheral control.
2. The Interrupt function for the peripheral control is turned off.
3. The central processor is initialized.

INTERNAL INTERRUPT

An internal interrupt signal is generated only by a Type 1201, 1251, or 2201 processor equipped with the Storage Protect Feature or a Type 4201 processor equipped with the Extended Multiprogramming and 8-Bit Transfer Feature and is caused by a "violation" of storage protection. (The nature of storage protect violations — internal interrupt address violation, op code violation, etc. — is described in Appendix E.) Processor indicators are set by the internal interrupt signal to show the cause (e.g., op code violation) and the type (internal) of interruption. These indicators can be stored and then tested by programmed instruction as described further in this appendix.

The processor reacts to the internal interrupt signal when the conditions described on page D-1 are present (i.e., the processor is in the RUN mode, is not in the external interrupt mode, is about to extract an op code, and is presently allocated a memory cycle) plus one additional condition: the processor must not only not be in the external interrupt mode but also must not be in the internal interrupt mode. Thus, the following levels of interrupt priority exist in the Type 1201, 1251, 2201, or 4201 processor.

1. If the processor is in the non-interrupt (standard) mode, normal program sequence can be interrupted by either an external or an internal source.
2. If the processor is in the internal interrupt mode, program sequence can be interrupted only by an external interrupt source.
3. If the processor is in the external interrupt mode, program sequence can not be interrupted.[1]

[1] Interrupt signals generated by any or all of the three external sources (peripheral control, control panel or console, or Monitor Call instruction) may continue to occur while the processor is in the external interrupt mode. The priority in which the interrupts are accommodated is determined by the program (i.e., according to the programmer-established sequence of interrupt source tests).

The processor responds to an internal interrupt signal as follows:

1. The contents of SR and IIR are interchanged, and the program branches to the instruction whose op code address was previously stored in IIR.
2. The processor enters the internal interrupt mode.

Note that the status of the arithmetic, comparison, address mode, and trap mode indicators are not stored in AIR automatically when the processor responds to an internal interrupt signal. The storing (and subsequent restoring) of the contents of these indicators is the responsibility of the internal interrupt program.

INTERRUPT PROGRAMMING

Three of the four interrupt control instructions (pages 8-92 through 8-101) perform basic functions in an interrupt routine:

1. The Store Variant and Indicators instruction (SVI) stores two types of information: (a) information which must be preserved for subsequent return to the interrupted program (e.g., indicator settings, variant register contents,[1] etc.); and (b) information required to identify the interrupt source.
2. The Restore Variant and Indicators instruction (RVI) restores the pertinent information stored by the SVI instruction before returning to the interrupted program.
3. The Resume Normal Mode instruction (RNM) returns the processor to continue sequencing in the interrupted program, unless the sector bits of SR have been modified.

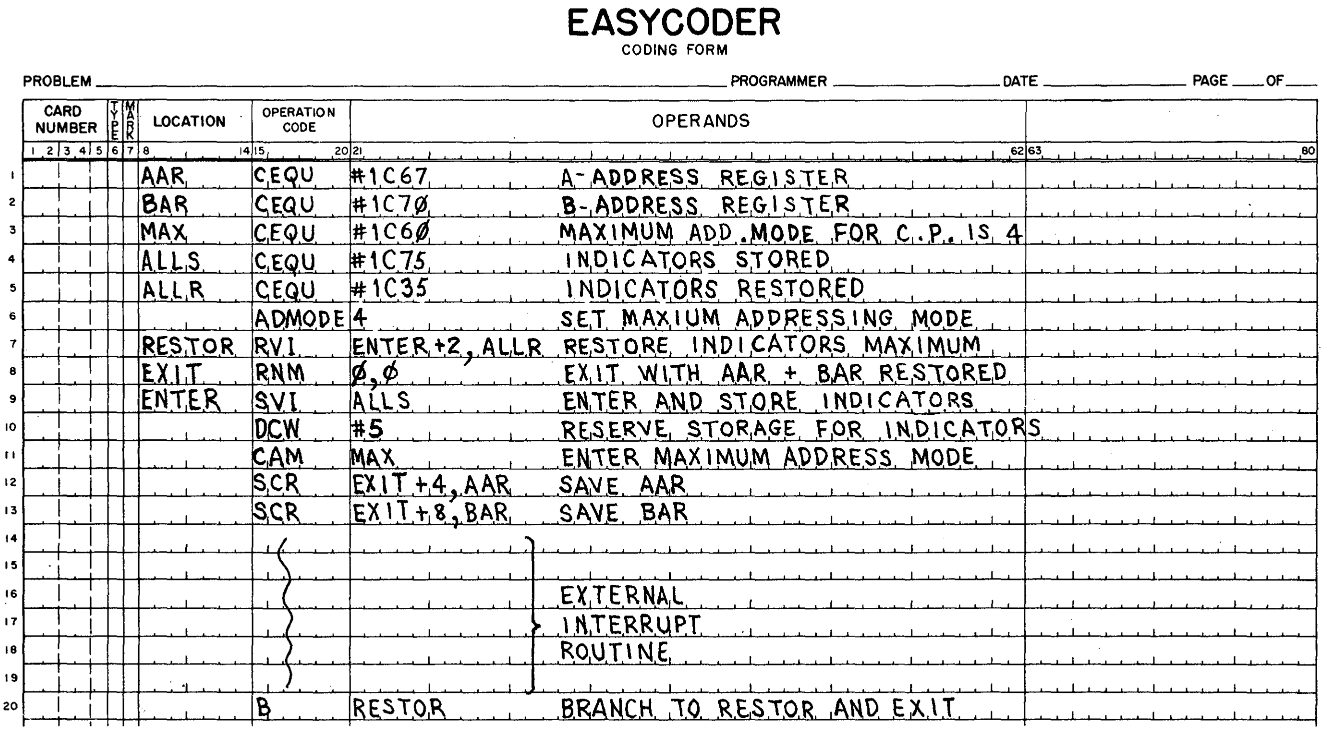
The fourth interrupt control instruction — Monitor Call (MC) — causes an external interruption and, therefore, is not coded in the interrupt routine itself.

Other instructions are required in the interrupt routine to store and exercise control over address register contents, as shown in Figures D-1 and D-2. The interrupt routines in these figures are assumed to be executed in the same sector as the interrupted program; if not, or if interrupt processing modifies the sector bits in SR, the appropriate sector bits must be stored upon entering the routine and restored when exiting.

For proper re-entry to the interrupted program, the same set of indicators stored by the SVI instruction should be restored by the RVI. Since the RVI instruction prepares the processor to re-enter the interrupted program, it should be followed immediately by the RNM instruction. Note that the A- and B-address register settings at the time of the interrupt should also be restored before re-entering the interrupted program. The external interrupt coding shown in Figure D-1 exploits the ability to restore the address registers automatically by storing their

[1] No means for storing the variant register contents is provided in the Type 201 and 201-1 processors. Therefore, when interrupt programming is used with these processors (optional on Type 201), variant characters must not be chained.

contents in the address fields of the RNM instruction. This technique requires that variant bit V_2 of the RVI instruction (see page 8-95) be a zero in order to ensure that the RNM instruction is executed in the maximum address mode of the machine. In an internal interrupt routine, on the other hand, the indicators associated with the V_2 must be stored and restored by the SVI/ RVI instructions. Therefore, since the address mode of executing the RNM instruction may not be maximum, the address fields of this instruction must not be coded. Instead, the address register settings must be stored in memory and restored by means of LCR instructions, as shown in Figure D-2.



EASYCODER
CODING FORM

PROBLEM ____ PROGRAMMER ____ DATE ____ PAGE __ OF __

	CARD NUMBER (1-5)	TYPE (6)	MARK (7)	LOCATION (8-14)	OPERATION CODE (15-20)	OPERANDS (21-62)	(63-80)
1				AAR	CEQU	#1C67 A-ADDRESS REGISTER	
2				BAR	CEQU	#1C7Ø B-ADDRESS REGISTER	
3				MAX	CEQU	#1C6Ø MAXIMUM ADD.MODE FOR C.P. IS 4	
4				ALLS	CEQU	#1C75 INDICATORS STORED	
5				ALLR	CEQU	#1C35 INDICATORS RESTORED	
6					ADMODE	4 SET MAXIUM ADDRESSING MODE	
7				RESTOR	RVI	ENTER+2,ALLR RESTORE INDICATORS MAXIMUM	
8				EXIT	RNM	Ø,Ø EXIT WITH AAR + BAR RESTORED	
9				ENTER	SVI	ALLS ENTER AND STORE INDICATORS	
10					DCW	#5 RESERVE STORAGE FOR INDICATOR	S
11					CAM	MAX ENTER MAXIMUM ADDRESS MODE	
12					SCR	EXIT+4,AAR SAVE AAR	
13					SCR	EXIT+8,BAR SAVE BAR	
14							
15							
16						EXTERNAL	
17						INTERRUPT	
18						ROUTINE	
19							
20					B	RESTOR BRANCH TO RESTOR AND EXIT	

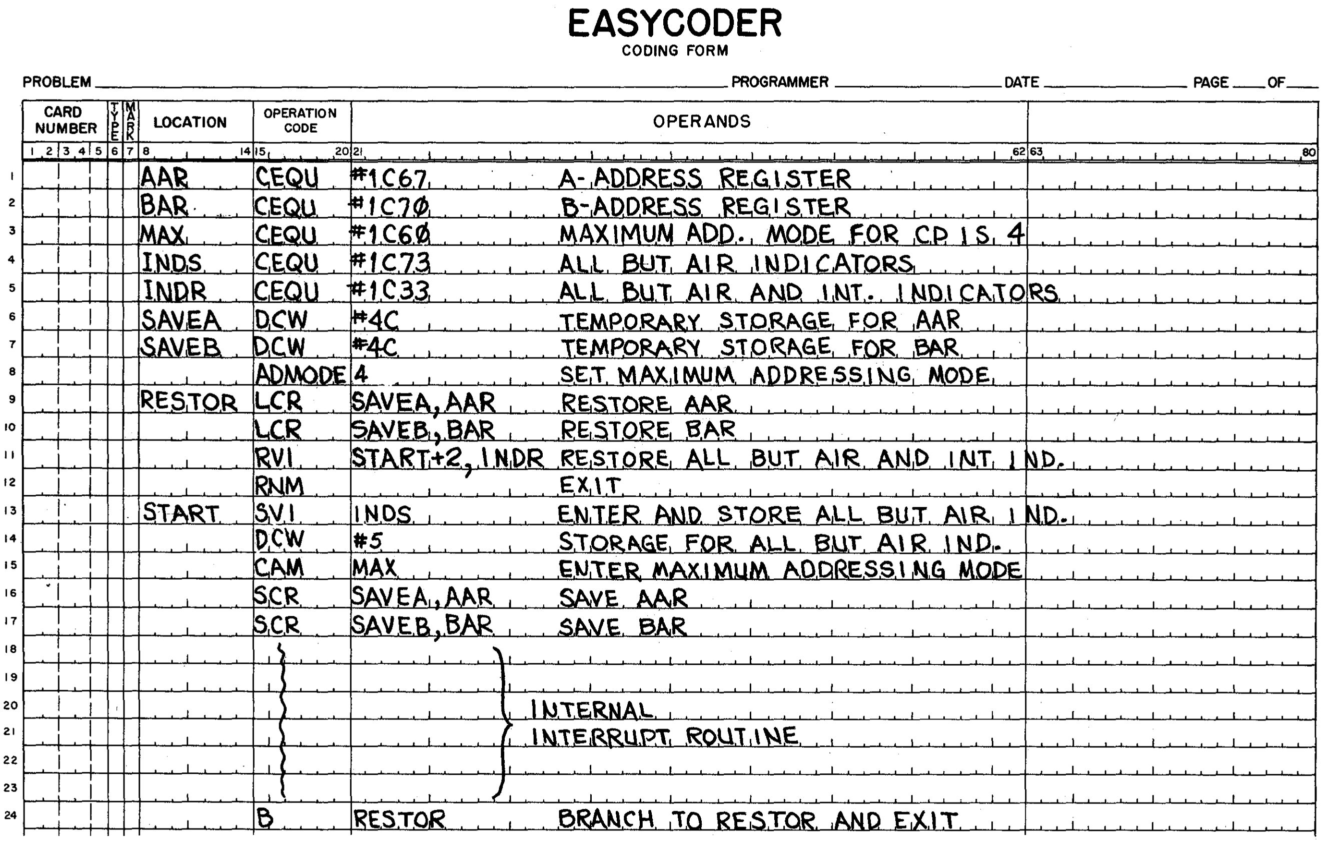
Figure D-1. Sample Coding For External Interrupt Routine

The first example (see Figure D-1) shows the initial and final coding to be used in an external interrupt routine. It is assumed that the address of the location tagged ENTER was previously stored in EIR, so that the presence of an external interrupt signal results in the automatic branch to the location tagged ENTER. It is assumed that the four-character addressing mode is the maximum addressing mode of the processor for which this routine is written.

NOTE: If the interrupt routine is not in the maximum addressing mode prior to branching to the location tagged RESTOR, a Change Addressing Mode instruction — CAM/MAX — must precede the RVI instruction so that the complete contents of any necessary control memory locations may be restored.

Figure D-2 shows the initial and final coding written for an internal interrupt routine. It is assumed that the address of the location tagged START was previously stored in IIR and that the maximim addressing mode of the processor is the four-character mode.

The initial and concluding instructions in an internal routine are similar to those in an external interrupt routine, except that the SVI instruction must store the indicators associated with bit V_2 and must not store the contents of the auxiliary indicators register (AIR). All other pertinent indicators are stored by the SVI instruction and are subsequently restored by the RVI instruction at the conclusion of the routine.



EASYCODER
CODING FORM

PROBLEM ______ PROGRAMMER ______ DATE ______ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	

```
AAR     CEQU    #1C67          A-ADDRESS REGISTER
BAR     CEQU    #1C7Ø          B-ADDRESS REGISTER
MAX     CEQU    #1C6Ø          MAXIMUM ADD. MODE FOR CP IS 4
INDS    CEQU    #1C73          ALL BUT AIR INDICATORS
INDR    CEQU    #1C33          ALL BUT AIR AND INT. INDICATORS
SAVEA   DCW     #4C            TEMPORARY STORAGE FOR AAR
SAVEB   DCW     #4C            TEMPORARY STORAGE FOR BAR
        ADMODE  4              SET MAXIMUM ADDRESSING MODE
RESTOR  LCR     SAVEA,AAR      RESTORE AAR
        LCR     SAVEB,BAR      RESTORE BAR
        RVI     START+2,INDR   RESTORE ALL BUT AIR AND INT IND.
        RNM                    EXIT
START   SVI     INDS           ENTER AND STORE ALL BUT AIR IND.
        DCW     #5             STORAGE FOR ALL BUT AIR IND.
        CAM     MAX            ENTER MAXIMUM ADDRESSING MODE
        SCR     SAVEA,AAR      SAVE AAR
        SCR     SAVEB,BAR      SAVE BAR
         {                 }
         {                 }
         {                 }  INTERNAL
         {                 }  INTERRUPT ROUTINE
         {                 }
         {                 }
        B       RESTOR         BRANCH TO RESTOR AND EXIT
```

Figure D-2. Sample Coding for Internal Interrupt Routine

PERIPHERAL CONTROL INTERRUPT

This description pertains to most Series 200 peripheral controls; exceptions are noted in the various hardware manuals describing individual peripheral devices.

Generally, a peripheral control's interrupt facility includes two interrelated functions: the Allow function and the Interrupt function. Certain controls have more than one set of functions (e.g., two sets for disk controls, but one set for magnetic tape controls). When a peripheral control becomes ready to accept a PDT instruction (i.e., reaches a "not-busy" status), it transmits a signal to turn on the Interrupt function, but this signal must be complemented by one from the Allow function (turned on by a PCB instruction) in order to complete the interrupt signal for transmission to the central processor (see Figure D-3).[1] When the Interrupt function is turned on, the interrupt signal is repeated continuously until the central processor is interrupted or the signal is turned off.

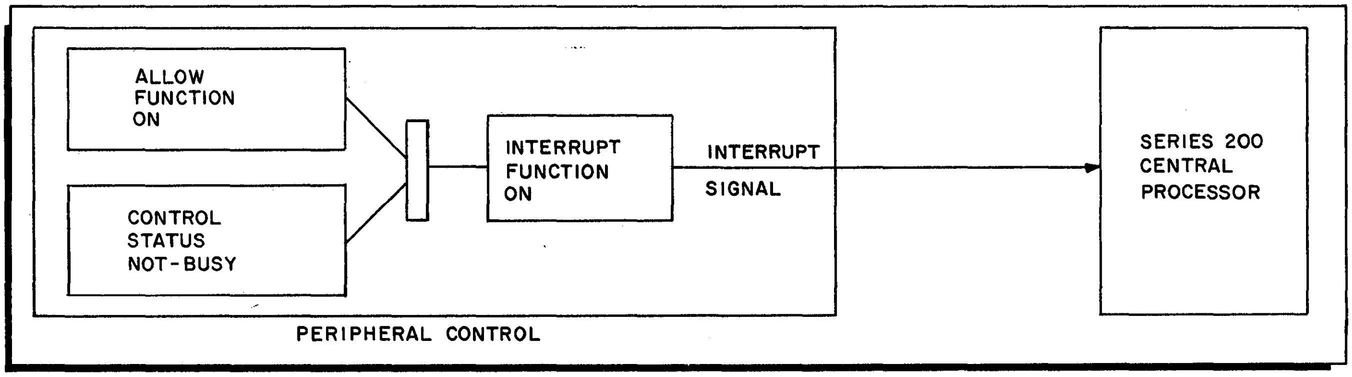


Figure D-3. Interrupt Signal Generated by Peripheral Control

The interrupt facility for a peripheral control can be activated or deactivated simply by turning the Allow function on or off, respectively. If the Allow function is off at the time the peripheral control becomes not busy and all error information is stored, the interrupt signal can be neither completed nor transmitted. Another method of inhibiting the interrupt facility is to turn off the Interrupt function; this function will not be turned on again until the control completes another PDT instruction. Note that if an interrupt has occurred and the Allow function has then been turned off, the Allow function should not be turned on again until either the Interrupt function has been turned off or a PDT instruction has been initiated by the control; otherwise, an interrupt occurs immediately.

There are various methods of turning the Allow or Interrupt function on or off. The Allow function can be turned on or off by a PCB instruction; similarly, the Interrupt function can be tested or turned off by a PCB instruction. Also, when the peripheral control receives a PDT instruction, its Interrupt function is turned off automatically; at completion of the PDT, a pulse is sent to turn on the Interrupt function. In any situation, both functions are turned off by initializing the central processor.

Specific PCB C3 characters for individual controls are listed in Tables 8-34, through 8-36. The C3 character in a PCB instruction may be used wither to control or to test the status of a

[1] This activity does not apply where there is no Allow function, as in the case of the Manual Interrupt function in a Type 220-3 Console connected to a Type 201 or 201-1 processor.

peripheral control's interrupt facility. The general formats of the C3 characters relating to interrupt control and test are:

1110x0 - Turn off the Allow function

1110x1 - Turn on the Allow function.

1111x0 - Turn off the Interrupt function.

1111x1 - Branch to A if the Interrupt function is on.

The 2-bit, shown here as x, is normally zero if the control being addressed contains only one set of Interrupt/Allow functions. If two sets of functions are present, this bit is set to identify the particular set being tested or controlled. All of these C3 characters result in a branch to A if the device addressed is not operable. Table D-1 summarizes Interrupt/Allow control and test operations for most peripheral controls; exceptions are noted in individual device manuals.

More than one control character can be used to specify multiple control and/or test operations in a PCB instruction. However, care must be taken in the use of certain combinations of these characters. For example, it is entirely possible for an interrupt to occur between extractions of control characters. In such a case, if control characters for "Branch on Interrupt" and "Turn Off Interrupt" were specified (in that order), the Interrupt function might be turned off without being acknowledged.

Table D-1. Summary of Interrupt/Allow Function Control and Test Operations

	RESULTING EFFECTS	
CONTROL/TEST OPERATIONS	ALLOW FUNCTION	INTERRUPT FUNCTION
Manual		
INITIALIZE Button	Turned off	Turned off
Program - PCB Control Char.[1]		
70	Turned off	None
71	Turned on	None
74	None	Turned off
75	None	Branch to A if on
Peripheral Control		
Upon receipt of PDT	None	Turned off
When PDT completed	None	Turned on if Allow on

[1] All of these PCB control characters will result in a branch to A if the device addressed is not operable.

APPENDIX E

STORAGE PROTECT FEATURE

When the Type 1201/1251 or 2201 processor is equipped with the Storage Protect capability (Feature 1114 or 1117, respectively), the main memory can be logically divided into two distinct areas: a protected area and an unprotected (or "open") area. When storage protection is in effect, the contents of the protected area are shielded from unintentional interference by any program operating in the standard (non-interrupt) mode (whether residing in the protected or unprotected area). The protected area is specified as follows:

1. The programmer sets the lower boundary of the area with a Load Index/Barricade Register (LIB) instruction specifying the number of a 4, 096-character memory bank (see page 8-79). The LIB instruction places this number in the index/barricade register. The lower boundary of the protected area is the leftmost (lowest) core storage location within this bank.
2. The upper boundary of the protected area is always the highest location in main memory.

The loading of the index/barricade register merely sets the low-order boundary of the protected area. In order to put storage protection into effect, the following conditions must be present:

1. The programmer must have turned the protect indicator on by issuing a Restore Variant and Indicators (RVI) instruction specifying the protect indicator (see page 8-95).
2. The processor must be in the standard (non-interrupt) mode.

INDEX REGISTERS

The Storage Protect Feature provides the user with an additional 15 index registers (Y1 through Y15), which are located in the leftmost 60 locations of the 4, 096-character bank specified by the current contents of the index/barricade register. Thus, these index registers are relocated whenever the contents of the index/barricade register are altered by an LIB instruction. These 15 registers are usable whenever the index/barricade register is loaded with a proper bank number and are not dependent upon whether storage protection is in effect or not. Instructions whose address portions are indexed by these registers must be assembled and executed in the four-character addressing mode. The high-order bit of the five-bit address modifier in a four-character address distinguishes index registers X1 through X15 from Y1 through Y15 (see page 4-14).

CENTRAL PROCESSOR MODES

As previously noted, the central processor can operate in any one of three modes:

1. The standard mode,

2. The external interrupt mode (see Appendix D), or
3. The internal interrupt mode.

Internal Interrupt

When storage protection is in effect (i.e., the protect indicator is on and the processor is operating in the standard mode), certain operations are defined as violations of that protection. These violations are discussed below. A violation causes a violation indicator to be set which, in turn, causes an internal interrupt to occur at the next opportunity. The "next opportunity" means that moment when all of the following conditions are present:

1. The processor is in the RUN mode (i.e., automatically executing stored-program instructions under the control of the sequence register),
2. The processor is about to extract an op code,
3. A memory cycle is allocated to the processor,
4. The processor is in the standard mode (i.e., not in external or internal interrupt mode), and
5. No peripheral or control panel interrupt signal is being received.

When an internal interrupt occurs, the contents of the sequence register and the internal interrupt register are interchanged and the central processor enters the internal interrupt mode. The status of the processor indicators are not stored automatically; therefore, the programmer must perform this function with a Store Variant and Indicators (SVI) instruction. The SVI instruction also clears the violation indicator so that an internal interrupt will not occur when a return is made to the standard mode. While in the internal interrupt mode, any external interrupt will cause the processor to switch to the external interrupt mode.

If an external interrupt occurs while the processor is in the internal interrupt mode, the 1-bit of the character stored by V5 of the SVI instruction indicates the condition. If it is desired to revert to the standard rather than the internal interrupt mode after servicing the external interrupt, this bit should be changed to 0 before executing the RVI instruction.

Note that three basic differences exist between the external interrupt mode and the internal interrupt mode:

1. A unique control memory location, the internal interrupt register (IIR), contains the address of the subroutine which services the internal interrupt,
2. The processor is subject to being interrupted by an external interrupt while still in the internal interrupt mode, but the reverse is not true,
3. No processor indicators are stored or altered (the address mode is not changed) upon entering the internal interrupt mode.

VIOLATIONS OF STORAGE PROTECTION

The following operations, which constitute violations of storage protection, fall into two general categories: address violations and op code violations.

1. An attempt to transfer information internally (i.e., not via a PDT instruction) to memory locations within the protected area. This includes any attempt to modify index registers Y1 through Y15. However, no violation occurs when information is transferred internally from the protected area or when the contents of the index registers are used in address modification. An internal transfer violation is detected when all of the following conditions are present:

   a. The bank and sector bits in the A- or B-address register following instruction extraction are equal to or greater than the corresponding bits stored in the index/barricade register,

   b. The protected location is addressed as a result location,

   c. The protect indicator is on,

   d. The program in control is operating in the standard mode, and

   e. The instruction is not a PDT.

   The above conditions are checked as the instruction is being executed. If all of these conditions are met, the internal interrupt address violation indicator is set, and the instruction proceeds to normal completion except that no information is transferred into memory (i.e., the write cycle is inhibited). The next opportunity for the internal interrupt to occur is at the extraction of the next op code. After the internal interrupt mode is entered, the internal interrupt register contains the address of the op code following the instruction which caused the violation, and the A- and B-address registers continue to increment or decrement, as appropriate.

2. An attempt to extract a PDT instruction (input or output) whose effective A address references a protected memory location. Since the PDT instruction is one of the operations normally prohibited when storage protection is in effect (see 4., below), the proceed indicator (see page E-5) must be set in order for the instruction to be extracted beyond the op code. Assuming that the proceed indicator is set, the starting address of the PDT operation is examined for address violation. Once it is determined that the effective A address references a protected address, no operation is performed (i.e., the specified read/write channel is not tested and the specified peripheral control is not addressed), the internal interrupt address violation indicator is set, the sequence register is advanced to the next op code, and an internal interrupt occurs.

   Note that a PDT instruction is checked for possible violation during the extraction phase, while a nonperipheral instruction is checked during its execution phase (see 1., above). If a PDT instruction passes this test during extraction, it is free to be executed and thereby cause data to be transferred. If the information being transferred extends into the protected area, no address violation is detected. To insure that this will not occur, the user must set a record mark immediately prior to the protected area.[1]

   As mentioned previously, storage protection (and the checking functions related to it) are in effect only when the processor is operating in the standard

[1] If communication devices are being used, two consecutive locations should contain record marks.

mode. However, violations of the protected area by PDT instructions executed in either of the two interrupt modes can be detected if the proceed indicator is set on (see page E-5).

3. An attempt to read from a main memory location whose address is greater than the main memory capacity actually present in the machine but within the addressing capacity of the memory address register.[1] Such an addressing attempt results in a parity error which normally causes the machine to halt. If storage protection is in effect and a parity error occurs, a check is made to determine whether the error occurred above the lower boundary of the protected area. If so, the storage protect hardware assumes that out-of-range addressing has been attempted,[2] no halt occurs, nor is data transferred; instead, the internal interrupt address violation indicator is set, instruction execution is prematurely terminated, and an internal interrupt occurs.

   An attempt to reference an address greater than the addressing capacity of the memory address register results in a memory wraparound.

4. An attempt to execute a privileged op code. A privileged op code is one which is (a) not defined for the Series 200; (b) not recognized on the particular processor; (c) an instruction format violation in any floating-point instruction; or (d) prohibited when storage protection is in effect. The privileged op codes in category (d) are:

   a. H (Halt)

   b. LCR (Load Control Registers)

   c. PDT (Peripheral Data Transfer)

   d. PCB (Peripheral Control and Branch)

   e. SVI (Store Variant and Indicators)

   f. RVI (Restore Variant and Indicators)

   g. RNM (Resume Normal Mode)

   h. LIB (Load Index/Barricade Register)

   The above op codes are "privileged" in the sense they are allowed to be executed in either of the interrupt modes but are prohibited in the standard mode while storage protection is in effect (one exception to this is discussed under "Proceed Indicator" below). Such op codes are categorized by the fact that they could possibly alter the monitor's knowledge of the status of the system or cause some action which is intolerable under certain conditions (e.g., a halt during transfer of data from a communications device). Since an undefined op code or one which is not installed on the user's processor would normally cause a halt due to a program check, such usage has the same effect as that of a privileged op code.

[1] For example, an MAR with 15 active bits can address up to 32,768 locations; an MAR with 16 active bits can address up to 65,536 locations. A 49,152-character memory would require 16 active bits, thus making it possible to store an address which is beyond the actual memory size.

[2] The final responsibility for checking whether the parity check actually indicates out-of-range addressing rests with the programmer.

NOTE: Op code "00" is defined as an Internal Interrupt Call, and falls within the category of privileged op codes.

If a privileged op code is extracted when storage protection is in effect, the op code violation indicator is turned on, the sequence register is set back to the location of the op code, the operation is terminated, and an internal interrupt occurs. Once the internal interrupt mode is entered, the programmer has two choices: (1) if he wishes to execute the privileged instruction, he must set the proceed indicator (see below) and issue a Resume Normal Mode (RNM) command;[1] (2) if he wishes to bypass the privileged instruction, he must set the internal interrupt register to the location of the next sequential op code and issue a Resume Normal Mode instruction.[2]

PROCEED INDICATOR

The proceed indicator can be turned on by the Restore Variant and Indicators (RVI) instruction. Turning this indicator on permits the execution of one privileged instruction in the standard mode without op code checking or item-mark trapping being performed. The indicator is turned off following the extraction of any op code in the standard mode. It can also be turned off in either of the interrupt modes by a Store Variant and Indicators (SVI) instruction.

The proceed indicator can also be used to force the checking of the A address of a PDT instruction executed in either the internal or external interrupt mode. Thus, turning on this indicator prior to the extraction of a PDT instruction in a nonstandard (interrupt) mode results in the same address violation check as though it were extracted in the standard mode with storage protection in effect (see 2., page E-3). If the effective A address is found to reference a protected area, the actions described below are performed.

1. When the violation occurs in the internal interrupt mode:
   a. The internal interrupt address violation indicator is set.
   b. Further extraction of the instruction is not performed and the sequence register is set to the location of the next sequential op code.
   c. An internal interrupt does not occur since the processor is already in the internal interrupt mode. Instead, the condition of the internal interrupt address violation indicator must be tested by the programmer after he has stored the status of the indicator via an SVI instruction. The SVI instruction also clears the indicator so that it will not cause an internal interrupt to occur when the standard mode is entered later.
2. When the violation occurs in the external interrupt mode:
   a. The external interrupt address violation indicator is set.

[1] The instruction will still not be executed if it involves an address violation.

[2] If the internal interrupt register (which is currently set at the location of the privileged op code) is not advanced to the next op code, the return to normal mode results in the privileged op code again being extracted, thus causing an endless loop.

b. Further extraction of the instruction is not performed and the sequence register is set to the location of the next sequential op code.

c. An internal interrupt does not occur since this is impossible while in the external interrupt mode. Instead, the condition of the external interrupt address violation indicator must be tested by the programmer according to the method described in 1.c., above.

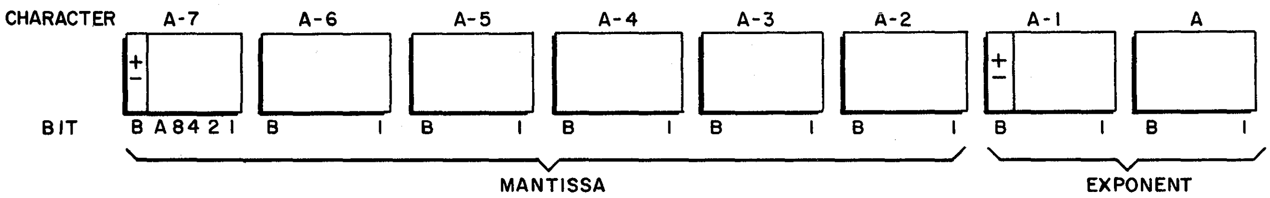
APPENDIX F

SCIENTIFIC UNIT FOR MODELS 1200, 1250, 2200, AND 4200

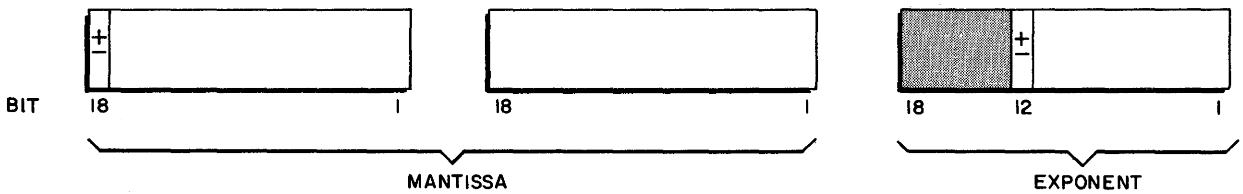
The scientific unit (Feature 1100A for the Type 1201, 1251, and 2201 processors, Feature 1101 for the Type 4201) provides a repertoire of 12 floating-point instructions, a binary mantissa shift instruction, and a binary integer multiply instruction.[1] This appendix is a programmer's working summary of both features, which are functionally identical;[2] additional information can be found in the hardware bulletin Scientific Unit for Models 1200/1250/2200, Feature 1100 (Order No. 126). Before referring to this appendix, the programmer should become familiar with the detailed functional and programming information contained in the hardware bulletin.

DATA FORMAT

The fixed-length floating-point word contains a 36-bit binary mantissa and 12-bit binary exponent and is capable of expressing numbers in the approximate range $\pm10^{\pm616}$



In control memory, a floating-point word may occupy any of the four floating-point accumulators. The accumulators are addressed as octal digits 0, 1, 2, and 3 in the floating-point instructions. Each accumulator comprises three specific 18-bit control memory registers. Only the low-order 12 bits of the rightmost register are used to express the exponent. (In the Type 4201 processor each accumulator comprises the low-order 18 bits in each of three specific 19-bit control memory registers.)



FLOATING-POINT REGISTERS

The four addresssable floating-point accumulators have the control memory addresses as shown on page F-2.

[1] None of these instructions are interpreted by Easycoder Assembler A, B, or C.

[2] A minor exception to this identity is described in connection with the Floating Divide instruction.

Accumulator Address	Control Memory Address (Operator's Control Panel Only)		
	High-Order Mantissa	Low-Order Mantissa	Exponent
0	43	42	41
1	47	46	45
2	53	52	51
3	57	56	55

NOTE: In program instructions, the floating-point accumulators may be addressed only via the octal digits 0, 1, 2, and 3 in the floating-point instructions. The instructions LCR and SCR must not be used to address these accumulators. At the control panel, the operator may address these locations with the addresses in the above table.

A normal zero, i.e., a floating-point word of 48 zeros, is stored in the "pseudo accumulator" for use as a floating-point operand. The pseudo accumulator, which is addressed by octal digit 7, may be used only as the source of a normal zero and not as the destination of a floating-point result.

The low-order result register (LOR) in the scientific unit may contain a low-order sum, difference, or product, or may contain the remainder of a division operation.

NOTE: Floating-point instructions do not disturb the contents of the variant register.

FLOATING-POINT INDICATORS

Exponent Overflow: Activated when a base-2 exponent exceeds +2047. The correct mantissa and an exponent which is 4096 less than the correct exponent are delivered. If an exponent is less than -2048, a normal zero is delivered automatically.

Divide Check: Activated when a divisor is equal to zero. This indicator causes termination of a division operation without accumulator alteration.

Multiply Overflow: Activated when the product of a Binary Integer Multiply instruction exceeds 24 bits in length. The low-order 24 bits are delivered.

AUTOMATIC FORMATTING IN ARITHMETIC OPERATIONS

Prenormalization: Mantissa of divisor (and dividend with Feature 1101) is normalized (left-shifted) with adjusted exponent.

Equalization: Mantissa of operand with smaller exponent is right-shifted until exponents are equal.

Postnormalization: Mantissa of result is normalized with adjusted exponent.

SYMBOLOGY

A: A address of the instruction.

B: B address of the instruction.

X: Floating-point accumulator addressed in the high-order three bits of an instruction variant (usually the source of an operand).

Y:	Floating-point accumulator addressed in the low-order three bits of an instruction variant (usually the destination of a result).
(A):	Floating-point word contained in the main memory field from location A through location A-7.
(X) or (Y):	Floating-point word contained in accumulator X or Y.
LOR:	Low-order result register.
(LOR):	Floating-point word contained in LOR.
A_p:	Previous setting of A-address register.
B_p:	Previous setting of B-address register.
D:	One if there is a two-bit overflow into LOR; otherwise zero.
JI:	Address of next instruction if branch occurs.
NXT:	Next sequential instruction.
N_n:	Number of bit positions shifted for automatic formatting.
N_1:	Number of binary ones in a multiplier.
N_s	Number of shifts.
[ ]:	"smallest integer greater than"
W:	Number of memory words used to store the data involved.
X-:	In the first variant of an instruction, only the high-order three bits specifying accumulator X are significant.
-Y:	In the first variant of an instruction, only the low-order three bits specifying accumulator Y are significant.
SP:	Single-precision.
DP:	Double-precision.
SR:	Sequence register.
Ni:	Number of characters in an instruction.
W_i:	Number of words in an instruction.

TIMING NOTES

All timings shown are based on the use of direct addressing. Three memory cycles should be added for each indexed address and one memory cycle should be added for each character extracted as a result of indirect addressing.

NOTES: Floating-point instructions do not disturb the contents of the variant register.

Table F-1. Summary of Scientific Instructions

FORMAT		OCTAL OP CODE	FUNCTION	REGISTERS AFTER OPERATION	TIMING (MEMORY CYCLES)		
					1201/1251	2201	4201
DATA MOVING INSTRUCTIONS							
STORE FLOATING ACCUMULATOR							
Memory to accumulator	FMA/A, X-, 00 or TAM/A, X-	07	(X) is stored in A through A-7. (X) is unaltered.	AAR: A-8 BAR: B_p	N_i+11	N_i+12	W_i+7
Accumulator to accumulator	FAA/XY, 00 or TAA/XY	06	(X) is loaded into Y. No normalization occurs	AAR: A_p BAR: B_p	7	8	W_i+4.5
LOAD FLOATING ACCUMULATOR							
Memory to accumulator	FMA/A, -Y, 02 or TMA/A, -Y	07	(A) is loaded into Y. No normalization occurs.	AAR: A-8 BAR: B_p	N_i+11	N_i+12	W_i+8
Accumulator to accumulator	FAA/XY, 02 or TAA/XY	06	(X) is loaded into Y. No normalization occurs.	AAR: A_p BAR: B_p	7	8	W_i+4.5
STORE LOW-ORDER RESULT							
Memory to accumulator	FMA/A, 00, 07 or TLM/A	07	(LOR) is stored in A through A-7. No normalization occurs.	AAR: A-8 BAR: B_p	N_i+10	N_i+11	W_i+8
Accumulator to accumulator	FAA/-Y, 07 or TLA/-Y	06	(LOR) is stored in Y. No normalization occurs.	AAR: A_p BAR: B_p	6	7	W_i+4
LOAD LOW-ORDER RESULT							
Memory to accumulator	FMA/A, 00, 01 or TML/A	07	(A) is loaded into LOR. No normalization occurs.	AAR: A-8 BAR: B_p	N_i+10	N_i+10	W_i+9
Accumulator to accumulator	FAA/X-, 01 or TAL/X-	06	(X) is loaded into LOR. No normalization occurs.	AAR: A_p BAR: B_p	6	6	W_i+4
FLOATING POINT ARITHMETIC INSTRUCTIONS							
FLOATING ADD							
Memory to accumulator	FMA/A, XY, 10 or AMA/A, XY	07	(A) is added to (X) and the sum is stored in Y. Indicator: Exponent overflow. Formatting: Equalization, postnormalization.	AAR: A-8 BAR: B_p LOR: Low-order result of operation. Sign bit = 0. Exponent = high-order exponent minus 35.	N_i+13+ $[N_n/6]$	N_i+13+$[N_n/4]$	W_i+13+N_n/6
Accumulator to accumulator	FAA/XY, 10 or AAA/XY	06	(X) is added to (Y) and the sum is stored in Y. Indicator: Exponent overflow. Formatting: Equalization, postnormalization.	AAR: A_p BAR: B_p LOR: Low-order result of operation. Sign bit = 0. Exponent = high-order exponent minus 35.	10+$[N_n/6]$	10+$[N_n/4]$	W_i+8+N_n/6
FLOATING SUBTRACT							
Memory to accumulator	FMA/A, XY, 11 or SMA/A, XY	07	Twos complement of (A) is added to (X) and the result is stored in Y. Indicator: Exponent overflow. Formatting: Equalization, postnormalization.	AAR: A-8 BAR: B_p LOR: Low-order result of operation. Sign bit = 0. Exponent = high-order exponent minus 35.	N_i+13+$[N_n/6]$	N_i+13+$[N_n/4]$	W_i+13+N_n/6
Accumulator to accumulator	FAA/XY, 11 or SAA/XY	06	Twos complement of (Y) is added to (X) and the result is stored in Y. Indicator: Exponent overflow. Formatting: Equalization, postnormalization.	AAR: A_p BAR: B_p LOR: Low-order result of operation. Sign bit = 0. Exponent = high-order exponent minus 35.	10+$[N_n/6]$	10+$[N_n/4]$	W_i+8+N_n/6
FLOATING MULTIPLY							
Memory to accumulator	FMA/A, XY, 13 or MAM/A, XY	07	(X) is multiplied by (A). The high-order product is stored in Y; the low-order product is stored in LOR. Indicator: Exponent overflow. Formatting: Postnormalization.	AAR: A-8 BAR: B_p LOR: Low-order product. Sign bit = 0. Exponent = high-order exponent minus 35.	N_i+18+$[N_1/6]$ +$[N_n/6]$	N_i+21+$[N_1/4]$ +$[N_n/4]$	Max=W_i+26+ +N_n/6 Min =W_i+18.5+ N_n/6
Accumulator to accumulator	FAA/XY, 13 or MAA/XY	06	(X) is multiplied by (Y). The high-order product is stored in Y; the low-order product is stored in LOR. Indicator: Exponent overflow. Formatting: Postnormalization.	AAR: A_p BAR: B_p LOR: Low-order product. Sign bit = 0. Exponent = high-order exponent minus 35.	15+$[N_1/6]$+ $[N_n/6]$	19+$[N_1/4]$+ $[N_n/4]$	Max=W_i+21.0+ N_n/6 Min=W_i+13.5+ N_n/6

Table F-1 (cont). Summary of Scientific Instructions

FORMAT		OCTAL OP CODE	FUNCTION	REGISTERS AFTER OPERATION	TIMING (MEMORY CYCLES)		
					1201/1251	2201	[illegible]
FLOATING POINT ARITHMETIC INSTRUCTIONS (cont)							
FLOATING DIVIDE							
Memory to accumulator	FMA/A, XY, 12 or DMA/A, XY	07	(A) is divided by (X). The quotient is stored in Y; the remainder is stored in LOR. Indicators: Exponent overflow, divide check. Formatting: Prenormalization of divisor (and of dividend with Feature 1101), postnormalization of quotient.	AAR: A-8 BAR: B_p LOR: Remainder. Sign = sign of dividend. Exponent = exponent of normalized dividend minus 35, and plus one if the absolute value of the dividend mantissa is greater than the absolute value of the mantissa of the normalized divisor.	$N_i+25+[N_n/6]$	$N_i+31+[N_n/4]$	$Max=W_i+30.5+N_n/6$ $Min=W_i+22+N_n/6$
Accumulator to accumulator	FAA/XY, 12 or DAA/XY	06	(Y) is divided by (X). The quotient is stored in Y; the remainder is stored in LOR. Indicators: Exponent overflow, divide check. Formatting: Prenormalization of divisor (and of dividend with Feature 1101), postnormalization of quotient.	AAR: A_p BAR: B_p LOR: Remainder. Sign = sign of dividend. Exponent = exponent of normalized dividend minus 35, and plus one if the absolute value of the dividend mantissa is greater than the absolute value of the mantissa of the normalized divisor.	$21+[N_n/6]$	$29+[N_n/4]$	$Max=W_i+25.5+N_n/6$ $Min=W_i+17+N_n/6$
CONVERSION INSTRUCTIONS							
DECIMAL TO BINARY	FMA/A, -Y, 03 or DTB/A, -Y	07	The 11-character signed decimal integer whose low-order character is A is converted to a 36-bit binary integer. The binary integer is stored in the mantissa portion of Y; the exponent of (Y) is set to +35. One- or two-bit mantissa overflow is possible. If mantissa overflow occurs, the low-order one or two bits are shifted into LOR. Y then contains the high-order result of conversion, with an exponent of 36 or 37. Normalization only occurs with overflow.	AAR: A-11 BAR: B_p LOR: Low-order result of conversion, Sign bit = 0. Exponent = high-order exponent minus 35.	N_i+20+D	N_i+24	$W_i+17.5$
BINARY TO DECIMAL	FMA/A, X-, 06 or BTD/A, X-	07	The mantissa portion of (X) is converted from a binary integer to a signed decimal integer. The decimal integer is stored in the 11-character main memory field whose low-order character is location A. The exponent portion of (X) is ignored and unaltered.	AAR: A-11 BAR: B_p	N_i+21	N_i+24	W_i+12+W
CONTROL INSTRUCTIONS							
FLOATING TEST AND BRANCH ON ACCUMULATOR CONDITION	FMA/A, XC, 04 or FBA/A, XC	07	The mantissa portion of (X) is tested for the condition specified by C, the low-order octal digit of variant 1. C=0 no branch C=1 (X) = 0 C=2 (X) < 0 C=3 (X) ≤ 0 C=4 (X) > 0 C=5 (X) ≥ 0 C=6 (X) ≠ 0 C=7 unconditional branch If the condition specified by C is satisfied, program control branches to location A. NOTE: (X) must be normalized.	AAR: A BAR: B_p NO BRANCH NXT BRANCH SR: NXT NO BRANCH JI(A) BRANCH	N_i+3 (NO BRANCH) N_i+4 (BRANCH)	N_i+3 N_i+5	$W_i+4.3$ W_i+5

Table F-1 (cont). Summary of Scientific Instructions

FORMAT	OCTAL OP CODE	FUNCTION	REGISTERS AFTER OPERATION	TIMING (MEMORY CYCLES)		
				1201/1251	2201	4201
CONTROL INSTRUCTIONS (cont)						
FLOATING TEST AND BRANCH ON INDICATOR FMA/A, 0D, 05 or FBI/A, 0D	07	The indicators specified by D, the low-order octal digit of variant 1, are tested. If any of the indicators is set, control branches to location A. D=0 no branch D=1 Multiply overflow D=2 Exponent overflow D=3 Exponent or multiply overflow D=4 Divide check D=5 Divide check or multiply overflow D=6 Divide check or exponent overflow D=7 Divide check, exponent overflow, or multiply overflow. NOTE: All indicators tested are reset.	AAR: A BAR: NXT BRANCH B_p NO BRANCH SR: NXT NO BRANCH JI(A) BRANCH	N_i+2 (NO BRANCH) N_i+3 (BRANCH)	N_i+2 N_i+4	W_i+3.8 W_i+4.3
BINARY MANTISSA SHIFT BMS/XM, V	04	If single-precision, the mantissa of (X) is shifted in the mode specified by M, the low-order octal digit of the first variant. If double-precision, the mantissas of (X) and (LOR) are shifted. The second variant V (0≤V≤63) specifies the number of positions by which bits are shifted. M=0 left, SP, rotate (end around) M=4 left, SP, arithmetic M=2 left, DP, rotate M=6 left, DP, arithmetic M=1 right, SP, rotate M=5 right, SP, arithmetic M=3 right, DP, rotate M=7 right, DP, arithmetic NOTE: The exponents of (X) and (LOR) are set to zero. In an arithmetic shift, the signs of the mantissas of (X) and (LOR) are preserved.	AAR: A_p BAR: B_p	$7+[N_s/6]$	$8+[N_s/4]$	$W_i+4.5+N_s/6$
BINARY INTEGER ARITHMETIC INSTRUCTIONS						
BINARY INTEGER MULTIPLY BIM/A, B	05	The four-character fields in memory whose low-order characters are A and B are treated as 24-bit binary integers. The integers are multiplied together; the product is stored in the field specified by the B address. Indicator: Multiply overflow.	AAR: A-4 BAR: B-4 LOR: unspecified	$N_i+21+[N_l/6]$	$N_i+23+[N_l/4]$	Max=W_i+21.5 Min=W_i+16.5

APPENDIX G

EXTENDED MULTIPROGRAMMING AND 8-BIT TRANSFER FOR MODELS 1200, 1250, 2200, and 4200

The extended multiprogramming and 8-bit transfer capability is available as Feature 1120, 1121, and 1118 on the Models 1200/1250, 2200, and 4200, respectively. The models 1200/1250 and 2200 must be equipped with the Storage Protect Features (1114 and 1117, respectively) before Features 1120 and 1121 can be added.

Extended multiprogramming provides a processor with five basic capabilities required in a multiprogramming environment and one feature required for upward compatibility. These are:

1. Base relocation,
2. Storage protection with base relocation,
3. Interrupt masking,
4. Instruction timeout,
5. 8-bit transfer capability, and
6. Privileged SCR Instructions.

STORAGE PROTECTION WITH BASE RELOCATION

In a processor equipped with extended multiprogramming, storage protection operates in either of two ways: with or without base relocation. Storage protection without base relocation operates as described in Appendix E.

The storage protection offered by extended multiprogramming is made possible by using base relocation in conjunction with storage protection. Base relocation is in effect when the relocation indicator is set (via the SVI and RVI instructions) and the processor is in the standard (non-interrupt) mode.

Storage protection with base relocation places a barrier above and below the area of memory where the active program is to operate, to prevent it from altering the contents of the rest of memory. The lower barrier is specified by the contents of the base relocation register (BRR), which is loaded and stored via Load Index/Barricade Register (LIB) and Store Index/Barricade Register (SIB) instructions. When relocation is in effect, the BRR is loaded with the bank address of the lowest memory bank (4,096 characters) available to standard mode programs. The BRR is added to each processor memory address transmitted to memory by a standard mode program. This prevents a standard mode program from writing into a memory bank below that specified by the BRR. The upper barrier is specified by the contents of the index barricade register (IBR). When storage protection is in effect and an attempt is made to write into memory at an address greater than that stored in the IBR, a protection violation occurs resulting in an internal interrupt. The IBR contains the number of 4,096-character memory banks which are available to a program.

A monitor program keeps track of the locations of the various programs stored in memory and, via the settings of the BRR and the IBR, can relocate references to any number of 4,096-character banks of memory. Thus, while there may be any number of programs stored in memory, only one program is active at any one time and all other programs are protected from the active program when storage protection is in effect. When, as the result of an interrupt, the monitor program activates a different program, it simply alters the settings of the BRR and the IBR to make available a different portion of memory.

Since all memory references are relocated via the BRR when relocation is in effect, index registers X1 through X15 effectively reside in the 4,096-character bank of memory specified by the BRR. The location of index registers Y1 through Y15 is also dependent on the setting of the relocation indicator. When relocation is activated, the Y index registers are also located in the 4,096-character bank specified by the BRR, where they become identical to index registers X1 through X15. When relocation is in effect, each program stored, including the monitor program, has its own set of 15 index registers when it is the active program. The index registers always reside in the memory area occupied by the active program.

EXTERNAL INTERRUPT MASKING

Each input/output (I/O) sector has associated with it a 1-bit mask. This mask is stored and set by Store Variant and Indicators (SVI) and Restore Variant and Indicators (RVI) instructions, respectively. When the mask for a sector is a zero, interrupts from sources in that sector are accepted and processed in the manner specified in Appendix D. When the mask for a sector is a one, then interrupts are held until the mask is altered or the interrupt function is reset. Control panel and Monitor Call interrupts are never masked. Depression of the INITIALIZE button on the control panel causes all mask bits to be reset to zeros.

INSTRUCTION TIMEOUT

It is possible for an instruction in a program to enter an infinite extraction or execution loop which would prevent a monitor program from servicing an interrupt within a specified time. To prevent this from occurring, a timeout function is provided which allows a maximum time limit to be placed on the extraction and the execution of any one instruction when the processor is in the standard mode. This function guarantees that a monitor program will, at some specified time, regain control of the system.

The instruction timer is reset to zero and begins timing every time the processor starts to extract or execute a new instruction. If the timeout allow function is on, the protect indicator is set, and the processor is in the standard mode when the time interval elapses, then the instruction being extracted or executed is terminated and an internal interrupt occurs.

The timeout function is enabled by a timeout allow function which is set and reset by the SVI and RVI instructions. Refer to pages 8-92 and 8-95 for SVI and RVI instructions.

8-BIT TRANSFER CAPABILITY

This capability allows central processor Types 1201, 1251, 2201, and 4201 to transfer data between peripheral controls and memory in either 6- or 8-bit format, as specified in the Peripheral Data Transfer (PDT) instruction.

1. The 6-bit mode is the standard data transfer mode used in Series 200 central processors. In this mode, only data is transferred between memory and peripheral controls. Punctuation is preserved in memory.
2. The 8-bit mode is used in those applications where an 8-bit transfer is desired between the central processor and a peripheral control. In this mode of operation, data and punctuation are transferred between the central processor and peripheral controls. Record marks in memory do not terminate data transfer in this mode.

When in the 8-bit mode, the number of 8-bit character transfers to be performed is determined by a 3-character count field in the PDT instruction or by control characters associated with the PDT peripheral controls.

The high-order bit of the C3 control character in a PDT instruction is a multivariant bit which conditions the peripheral control in its interpretation of the remainder of the instruction. When this bit is a zero, all additional control characters beyond C3 are ignored by the control. When the high-order bit of C3 is a one, additional control characters are present and will be accepted by the peripheral control. In this case, the format of the PDT instruction becomes:

Op code/A address/C1, C2, C3, C4, C5, C6, C7.

Control character C4 is always present when the multivariant bit (bit 6 of C3) is a one. When the extended bit (bit 5 of C3) is a one, control characters C5, C6, and C7 are present. When the extended bit is a zero, control characters C5, C6, and C7 are ignored. The high-order bit of C4 determines the data transfer mode; one specifies 8-bit mode and a zero specifies 6-bit mode. Because 8-bit mode data transfers are not affected by record marks, data transfer is delimited by the setting of the extended bit in the C3 control character. If this bit is a zero, all data transfers previously terminated by a record mark are now terminated by transferring the number of characters specified in the record header area. If it is a one, all data transfers previously terminated by a record mark are now terminated by transferring the number of characters specified by the count field (C5, C6, and C7) of the PDT instruction.

PRIVILEGED SCR INSTRUCTION

When a processor is in the standard mode with the storage protection indicator ON and the proceed indicator OFF, the detection of an SCR instruction having a variant character of octal 00 through octal 37 will set the op code violation indicator and cause an internal interrupt to occur at the next opportunity.

The following status is specified at the conclusion of the trapped SCR instruction.

1. The internal interrupt register (IIR) contains the address of the privileged op code.
2. The A-address register (AAR) contains the address of the previous instruction.
3. The main memory locations specified by the A-address are undisturbed.
4. The variant register contains the variant character of the privileged SCR instruction.

All SCR instructions are identically executed if the proceed indicator is ON.

EXTENDED INPUT/OUTPUT CAPACITY FOR THE MODEL 4200

APPENDIX H

An extended input/output capacity for the Model 4200 is available as Features 1116, 4214A, 4214B, and 4215.

FEATURE 1116

Feature 1116 increases the peripheral flexibility of the Model 4200 by providing a third input/output sector. This feature includes eight additional read/write channels for a total of 16, and facilities which allow the permanent connection of 16 additional peripheral controls for a total of 48. With this expanded system, up to 16 read/write channels can be used simultaneously for data transfer operations.

Sector 3 handles up to four peripheral devices simultaneously and has a maximum data transfer rate of 333, 333 characters per second. Thus when feature 1116 is included, the I/O controller can accommodate a peak data transfer rate of 1, 333, 333 characters per second.

FEATURES 4214A and 4214B

Features 4214A[1] (Two Buffered I/O Sectors) and 4214B[2] (Two Additional Buffered I/O Sectors), provide the Model 4200 with buffered I/O sectors for those applications where additional compute time or a higher input/output transfer capability is required. When both features are included, sectors 1 and 3 remain unchanged but sector 2 is replaced with four buffered sectors. Each buffered sector has a data transfer rate of 500, 000 characters per second, can handle up to 6 peripheral devices simultaneously, and provides facilities to permanently attach up to 16 modate a data transfer rate of 2, 833, 333 characters per second and perform a total of 16 simultaneous input/output operations. In addition to increasing the I/O capability of the Model 4200, Features 4214A and 4214B reduce the usage of available memory cycles by the I/O controller. Consequently, the memory cycles saved are available to the central processor.

FEATURE 4215[1]

Feature 4215 (High-Speed Third Sector) increases the transfer rate of that sector to 1, 333, 333 characters per second. This allows connection of I/O peripheral devices with transfer rates exceeding 500, 000 characters per second to the third sector. When Feature 4215 is added

[1] Requires the installation of Feature 1116.

[2] Requires the installation of Feature 4214A.

to the system, the data transfer rate is 2,333,333 characters per second. When Feature 4215 is included as well as Features 4214A and 4214B, the I/O controller can accommodate a peak data transfer rate of 3,833,333 characters per second.

BUFFERED SECTORS

A single buffered sector is equipped with six 4-character buffers. Therefore, up to six devices operating concurrently are provided with a 4-character storage area. A buffer accumulates up to 4 characters of data before requiring access to main memory. The buffered sectors may be used without their buffer areas (direct mode) but this arrangement results in a slower data transfer rate. Table H-1 indicates whether or not a control/device can be connected to a buffered sector in either the buffered or in the direct mode.

In order to attain optimum system performance, sectors and their maximum data transfer rates should be taken into consideration before permanently connecting peripheral controls to particular sectors.

Table H-1. Controls/Devices Connectable to Buffered Sectors

Peripheral Control/Device	Buffered Mode	Direct Mode
Type 203 — Tape Controls (all)	Yes	No
Type 206 — High-Speed Printer Control	No	No
Type 206A — Printer Control for 822-3	No	No
Type 207 — Card Reader Control (for Type 227)	No	No
Type 208 — Card Punch Control (for Type 227)	No	No
Type 208-1 — Card Punch Control (for 224-1, -2, or 214-1).	Yes	Yes
Type 208-2 — Card Read/Punch Control (for 224-1, -2, or 214-2)	Yes	Yes
Type 209 — Paper Tape Reader and Control	No	Yes
Type 209-2		
Type 210 — Paper Tape Punch and Control	Yes	Yes
Type 212 — On-Line Adapter	No	No
Type 212-1 — Central Processor Adapter	No	Yes
Type 213-3 — Interval Timer	Yes	Yes
Type 213-4 — Time of Day Clock	Yes	No
Type 220-1, -2, -3 — Console	No	Yes
Type 222-1, -2, -3, -4, -5, -6 — Printer and Control	Yes	No
Type 223 — Card Reader and Control	Yes	No
Type 223-2 — Card Reader and Control	Yes	No
Type 229 — Printer and Control	Yes	No
Type 232 — MICR Reader-Sorter and Control	No	Yes
Type 233-2 — MICR Control	No	Yes
Type 234 — Plotter Control	Yes	Yes
Type 235 — Optical Journal Reader Control	Yes	Yes
Type 237 — Bill Feed Printer Control	No	No
Type 238 — Optical Reader Control	No	No
Type 257 — Control for 258, 259 Disk Pack Drives	No	No
Type 257-1 — Control for 258, 259 Disk Pack Drives (6-and 8-bit transfer)	No	No

Table H-1 (cont). Controls/Devices Connectable to Buffered Sectors

Peripheral Control/Device	Buffered Mode	Direct Mode
Type 257A — Control for 259A Disk Pack Drive	No	No
Type 257B — Control for 259B Disk Pack Drive	No	No
Type 257B-1— Control for 259B Disk Pack Drive (-6 and 8-bit transfer)	No	No
Type 260 — Control for 261 and 262 Disk Files	Yes	No
Type 260-1 — Control for 265, 266 High-Speed Drums	Yes	No
Type 260-2 — Control for 267 High-Speed Drums	No	No
Type 270A-1, -2, -3 — Random Access Drum Storage and Control	No	No
Type 281 — Single-Channel Communication Controls (all)	No	Yes
Type 286-1, -2, -3, -4, -5 — Multi-Channel Communication Controls (all)	No	No
Type 287 — AUTODIN Communication Control	No	Yes
Type 287-1 — USASCII AUTODIN Communication Control	No	Yes

COMPUTER-GENERATED INDEX

Honeywell

HONEYWELL
TECHNICAL PUBLICATIONS REMARKS FORM

TITLE: SERIES 200
PROGRAMMERS' REFERENCE MANUAL
(MODELS 200/1200/1250/2200/4200)

DATED: OCTOBER, 1968

FILE NO: 113.0005.0000.2-139

ERRORS NOTED IN PUBLICATION:

Fold

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION:

Fold

(Please Print)

FROM: NAME ______________________ DATE ____________

COMPANY ______________________

TITLE ______________________

ADDRESS ______________________

____

2837

FIRST CLASS
PERMIT NO. 39531
NEWTON HIGHLANDS
MASS.

BUSINESS REPLY MAIL
No postage stamp necessary if mailed in the United States
POSTAGE WILL BE PAID BY

HONEYWELL
151 NEEDHAM STREET
NEWTON HIGHLANDS, MASS. 02161

ATT'N: MARKETING INFORMATION SERVICES, MS 251

Honeywell