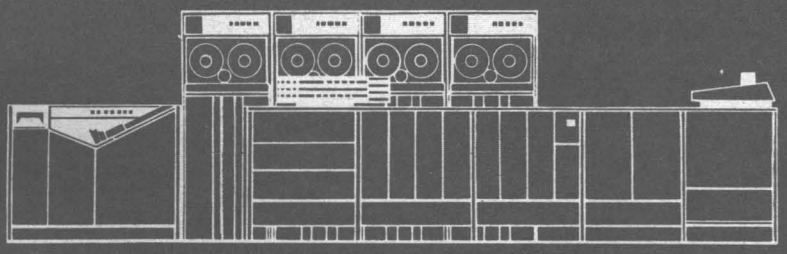
HONEYWELL SERIES 200

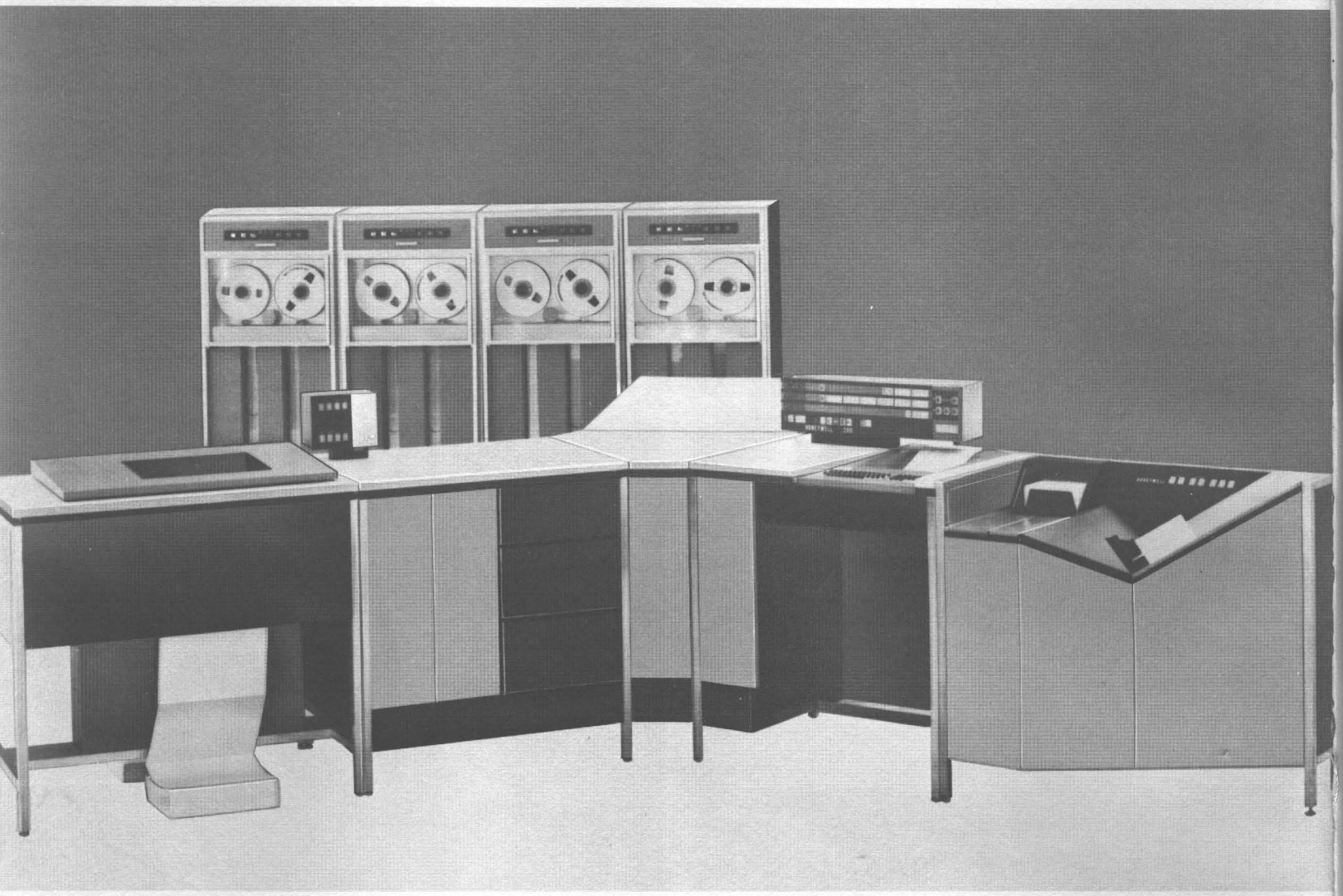
Models 200/1200/2200

PROGRAMMERS' REFERENCE MANUAL



Honeywell

ELECTRONIC DATA PROCESSING



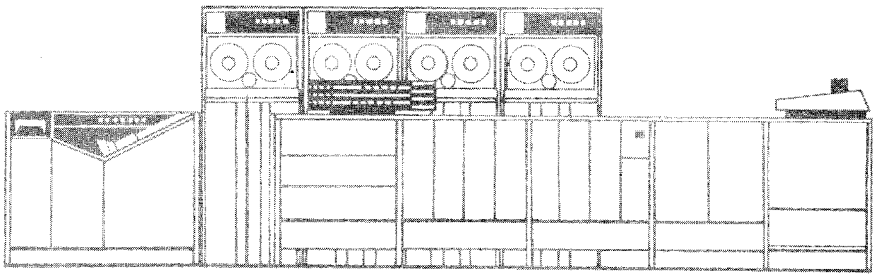
Printed in U.S.A.
WP-8324

113.0005.0000.00.00
10965

HONEYWELL SERIES 200

Models 200/1200/2200

PROGRAMMERS' REFERENCE MANUAL



FIRST EDITION
First Printing September, 1965

Honeywell
ELECTRONIC DATA PROCESSING

PRICE $4.50

Questions and comments regarding this manual should be addressed to:

Honeywell Electronic Data Processing
Information Services
60 Walnut Street
Wellesley Hills, Massachusetts 02181

PREFACE

The purpose of this text is to provide a detailed reference source containing:

1. a functional description of the Honeywell Series 200 Models 200, 1200, and 2200 and their components.
2. a definition of the Series 200 Assembly System (Easycoder).
3. a detailed explanation of machine operation codes.

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

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

The following publications are hereby superseded:

Honeywell 200 Programmers' Reference Manual (DSI-214),
Honeywell 2200 Programmers' Reference Manual (DSI-304),
Easycoder 8K Assembly Language (DSI-409), and
Easycoder 12K Assembly Language (DSI-313).

TABLE OF CONTENTS

TABLE OF CONTENTS (cont)

LIST OF ILLUSTRATIONS

LIST OF TABLES

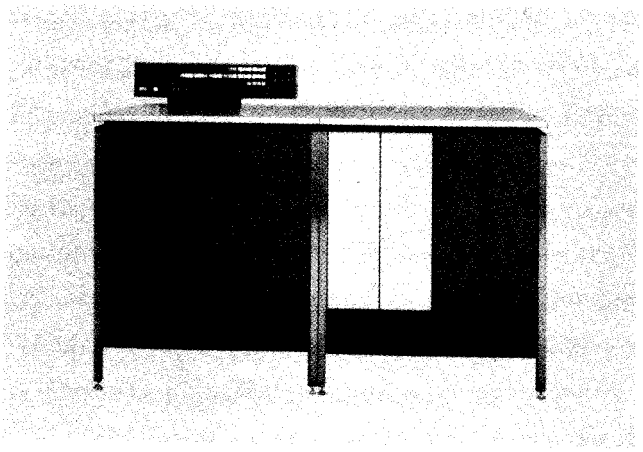
1

SERIES 200 COMPONENTS

Honeywell's Series 200 Data Processing System is a set of modularly designed, compatible models, three of which — the Models 200, 1200, and 2200 — 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 trunks available with any one 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, five central processors are described herein: the three processors of Model 200 (Types 201, 201-1, and 201-2); the Type 1201; and the Type 2201. 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.

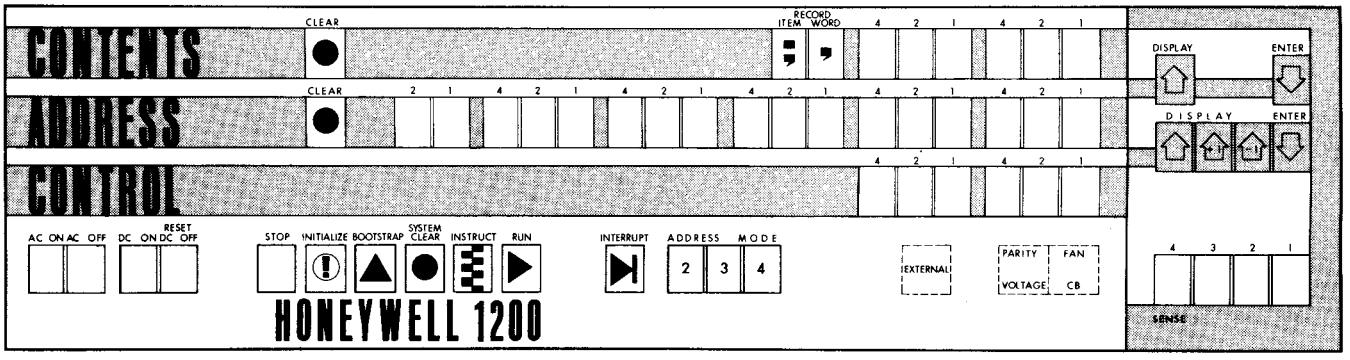


Figure 1-1. Type 1201 Control Panel

Another communication medium between the operator and the central processor is the Type 220 console, of which three 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 operator 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 220-2 and Type 220-3 consoles (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 1-17) and the additional SENSE switches used with the larger Series 200 processors.

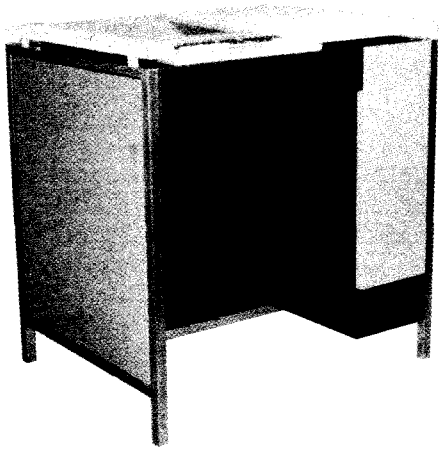


Figure 1-2. Type 220-1 Console

Figure 1-3. Type 220-2 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 when the processor is interrupted. Any one of four sources (see below) can "demand" access to the central processor by generating an interrupt signal; this signal turns on a central processor interrupt indicator. An automatic hardware response is made to this condition: information concerning the current status of the processor 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 interruption is automatic. When the stored service routine has been executed, control is returned to the main program at the point where the interruption occurred.

The four sources of processor interruption 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 1-6). 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.

4. Storage Protect Violation — The above-mentioned sources cause an external interrupt condition. When a processor contains the Storage Protect Feature (Types 1201 and 2201 only), an internal interrupt condition, caused by certain "violations" to storage protection, can also occur. Internal interruptions are of lower priority than external interruptions, so that a processor executing an external interrupt service routine cannot be interrupted by an internal interruption until the routine is completed. The nature of storage protect violations is described in Appendix E.

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 200/1200/2200 ranges from a minimum of 2,048 character locations (Types 201 and 201-1) to 262,144 locations (Type 2201). Figure 1-4 shows the modular main memory structures of the five 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 single character location). These speeds range from two microseconds to one microsecond for the five processors (see Figure 1-5).

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

Peripheral simultaneity is a key feature of Series 200 processors. Among the processors described in this manual, from three (Model 200 processors) to eight (Type 2201 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.

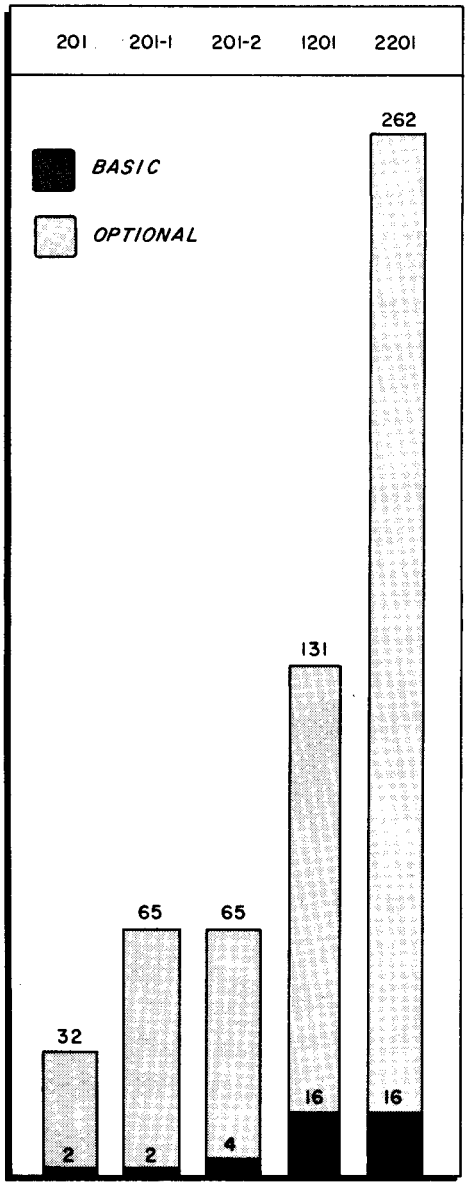


Figure 1-4. Main Memory Size

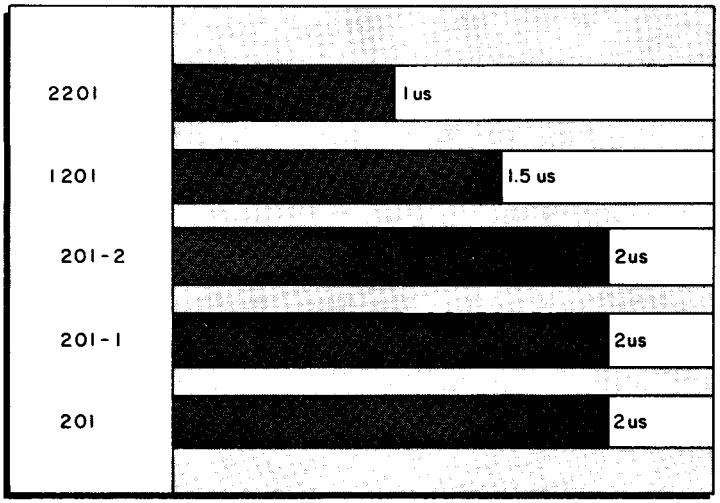


Figure 1-5. Main Memory Speed

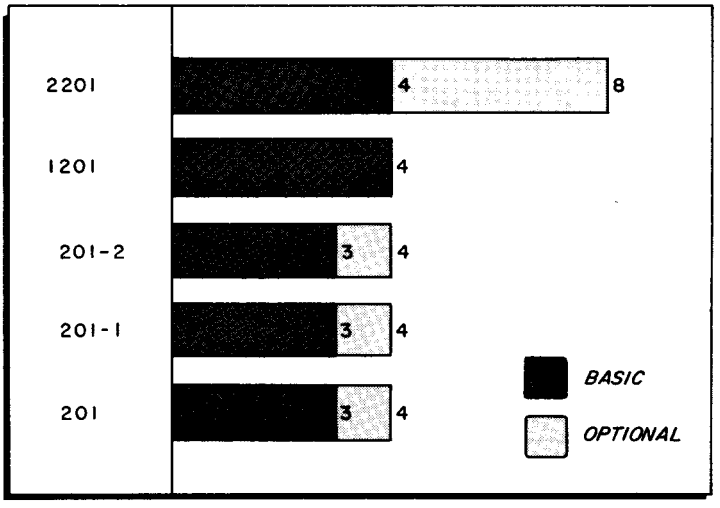
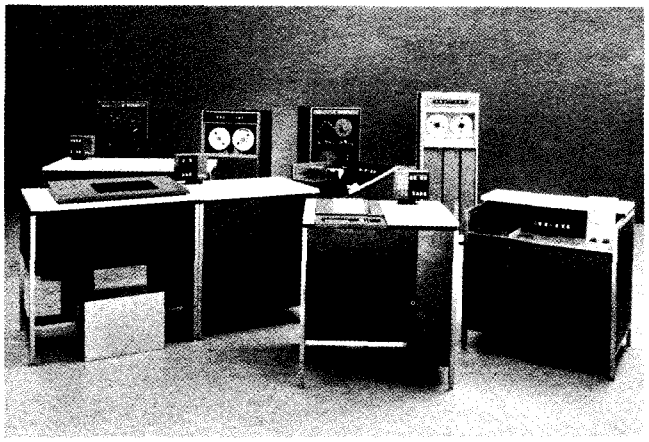


Figure 1-6. Peripheral Simultaneity

PERIPHERAL EQUIPMENT



The array of peripheral devices available with Series 200 processors includes over 40 units: punched card equipment, high-speed printers, magnetic tape units, paper tape equipment, magnetic tape strip mass memory units, random access drum units, and various data communication equipment. Also included are computer-to-computer adapters, an interval timer, a time of day clock, MICR reader/sorter controls, and peripheral switching units which provide extremely flexible Series 200 configurations.

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 8. 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 the following tables.

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, six different punched card units are offered: a card reader, three card punches, and two reader/punches. Table 1-1 lists the card devices available within Series 200. Note that a card device requires either one or two "I/O trunks," depending on the number of functions the device performs. The significance of the I/O trunk is explained on page 1-13.

Table 1-1. Series 200 Punched Card Equipment

Device		Data Transfer Rate	No. Devices Per Control	No. I/O Trunks Required by Control
Type	Function			
223	Card Reader	800 cards/minute	1	1
214-1	Card Punch	100-400 cards/minute	1	1
224-1	Card Punch	50-270 cards/minute	1	1
224-2	Card Punch	90-360 cards/minute	1	1
214-2	Card Reader/Punch	Read: 400 cards/minute Punch: 100-400 cards/minute	1	2
227	Card Reader/Punch	Read: 800 cards/minute Punch: 250 cards/minute	1	2

HIGH-SPEED PRINTERS

Five 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 data, suppress zeros, and insert identifying symbols in the data prior to printing.

Table 1-2. Series 200 High-Speed Printers

Type	Data Transfer Rate	No. Printers Per Control	No. I/O Trunks Required by Control
222-1 (96 print positions)	650-1,300 lines/minute	1	1
222-2 (108 print positions)	650-1,300 lines/minute	1	1
222-3 (120 or 132 print positions)	650-1,300 lines/minute	1	1
222-4 (120 or 132 print positions)	950-1,266 lines/minute	1	1
222-5 (120 print positions)	450 lines/minute	1	1

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 7,200 to 83,300 characters per second; three types of 3/4-inch tape units read/write from 32,000 to 88,800 characters per second.

Table 1-3. Series 200 Magnetic Tape Units

Type	Data Transfer Rate	No. Devices Per Control	No. I/O Trunks Required by Control
1/2-Inch Magnetic Tape Units			
204B-1 204B-2	7,200/20,000 characters/second	1-8	2
204B-3 204B-4	16,000/44,400 characters/second	1-8	2
204B-5	24,000/66,700 characters/second	1-8	2
204B-6	30,000/83,300 characters/second	1-8	2
204B-7	7,200/20,000/28,800 characters/second	1-8	2
204B-8	16,000/44,400/64,000 characters/second	1-8	2
204B-11 204B-12	13,300 characters/second	1-4	2
3/4-Inch Magnetic Tape Units			
204A-1	32,000 characters/second	1-4	2
204A-2	64,000 characters/second	1-4	2
204A-3	88,800 characters/second	1-4	2

MASS MEMORY FILE

Honeywell's reputation for reliable magnetic tape control is inherited by the new member of Series 200, the Mass Memory File. Three types of transports, varying in access time and capacity, use magnetic tape strips to store programs and data files and thereby complement the main memory storage capacities of the central processor (see Table 1-4). A single control can provide access to over two billion characters of stored information. Data transfer rate is 100,000 characters per second, and average access times are as low as 95 milliseconds.

Table 1-4. Series 200 Mass Memory File Units

Device		Data Transfer Rate	No. Devices Per Control	No. I/O Trunks Required by Control
Type	Function			
251	Magnetic Tape Strip Transport (15 million characters)	100,000 characters/second	1-8	2
252	Magnetic Tape Strip Transport (60 million characters)	100,000 characters/second	1-8	2
253	Magnetic Tape Strip Transport (300 million characters)	100,000 characters/second	1-8	2

RANDOM ACCESS DRUM FILE

The Series 200 drum file features a control unit which can direct from one to eight magnetic drums, each capable of storing over two million characters of information (see Table 1-5). Thus, a single drum file subsystem can have a total capacity of over 20 million characters. Any record stored on the drum can be accessed in 27.5 milliseconds (average) and can be transferred at the rate of 102,000 characters per second.

Table 1-5. Series 200 Magnetic Drum File Units

Device		Data Transfer Rate	No. Devices Per Control	No. I/O Trunks Required by Control
Type	Function			
270-1 through 270-8	Magnetic Drum (2.6 million characters)	102,000 characters/second	1-8	2

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 1-6).

Table 1-6. Series 200 Paper Tape Equipment

Device		Data Transfer Rate	No. Devices Per Control	No. I/O Trunks Required by Control
Type	Function			
209	Paper Tape Reader	600 characters/second	1	1
210	Paper Tape Punch	120 characters/second	1	1

DATA COMMUNICATION EQUIPMENT

The immediate and automatic response to an external interrupt source by the Series 200 processor was described previously (page 1-3). A common source of external interruption is the communication control, of which two different types are available in Series 200. 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. Both single-channel and multi-channel data communication controls are offered; these controls adapt themselves to a broad selection of lines, speeds, and terminal devices. One such terminal device is Honeywell's Data Station (see Table 1-7).

Table 1-7. Series 200 Data Communication Equipment

Device		Data Transfer Rate	No. Devices Per Control	No. I/O Trunks Required by Control
Type	Function			
		Communication Controls		
281	Single-Channel Control	Up to 5,100 characters/second	1 line	2
286	Multi-Channel Control	Up to 300 characters/second/line	1-63 lines	2
		Remote Terminal Device		
288-1	Data Station Central Control	120 characters/second	n/a	n/a
288-2	Data Station Central Control & Keyboard	120 characters/second	n/a	n/a
289-2	Data Station Page Printer & Keyboard	10 characters/second	n/a	n/a
289-3	Data Station Page Printer & Keyboard	40 characters/second	n/a	n/a
289-4	Data Station Paper Tape Reader	120 characters/second	n/a	n/a
289-5	Data Station Paper Tape Punch	120 characters/second	n/a	n/a
289-6A	Data Station Paper Tape Reader/Punch	50 characters/second	n/a	n/a

Table 1-7 (cont). Series 200 Data Communication Equipment

Device		Data Transfer Rate	No. Devices Per Control	No. I/O Trunks Required by Control
Type	Function			
Remote Terminal Device				
289-6B	Data Station Paper Tape Reader	50 characters/second	n/a	n/a
289-7	Data Station Card Reader	120 characters/second	n/a	n/a
289-8	Data Station Optical Bar Code Reader	50 characters/second	n/a	n/a

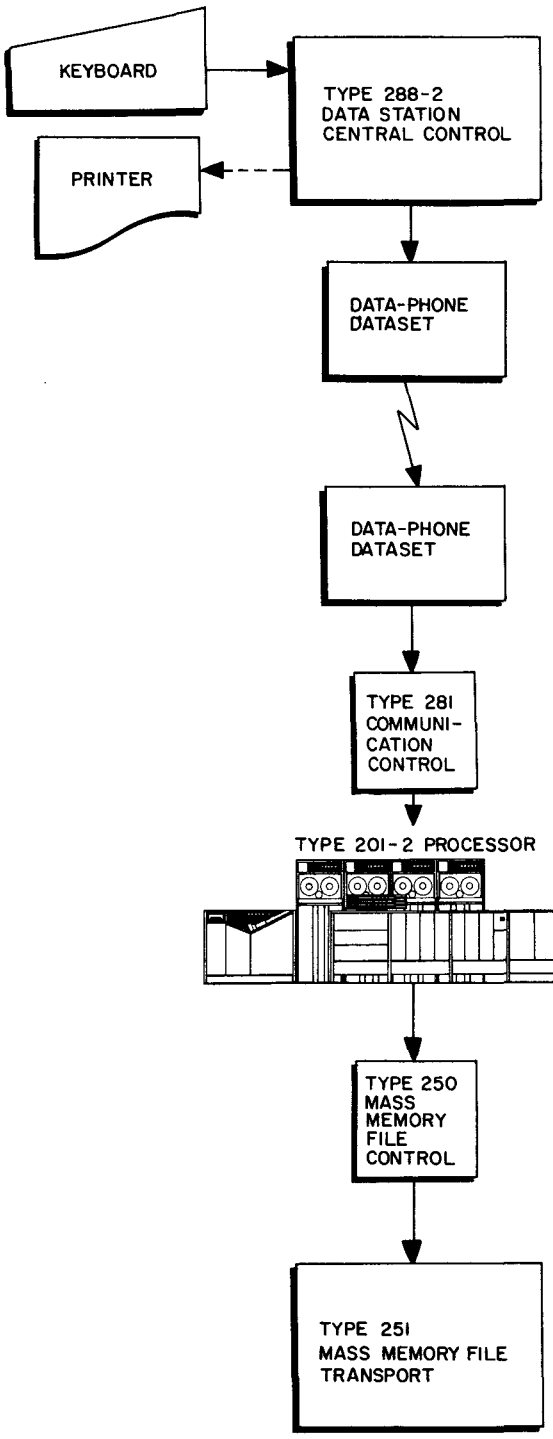
A major requirement of many communication networks (e.g., inquiry handling or message switching applications) is fast access to a stored file. Files may sometimes be stored in main memory, but for large files main memory storage is economically unfeasible. File storage units (i.e., the Mass Memory File, magnetic tape units, or drum file units) provide the answer to such mass storage 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 251 Mass Memory File transport; (3) a Type 281 communication control; (4) two DATA-PHONE 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.

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 2201 processor performs as many as eight simultaneous peripheral operations. 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

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



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 pre-assigned 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 251 Mass Memory File transport.
7. Type 250 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 pre-assigned 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

this is explained in Section 2. The data path used by the traffic control to transfer data is described below; Figure 1-8 illustrates the basic elements which form this data path.

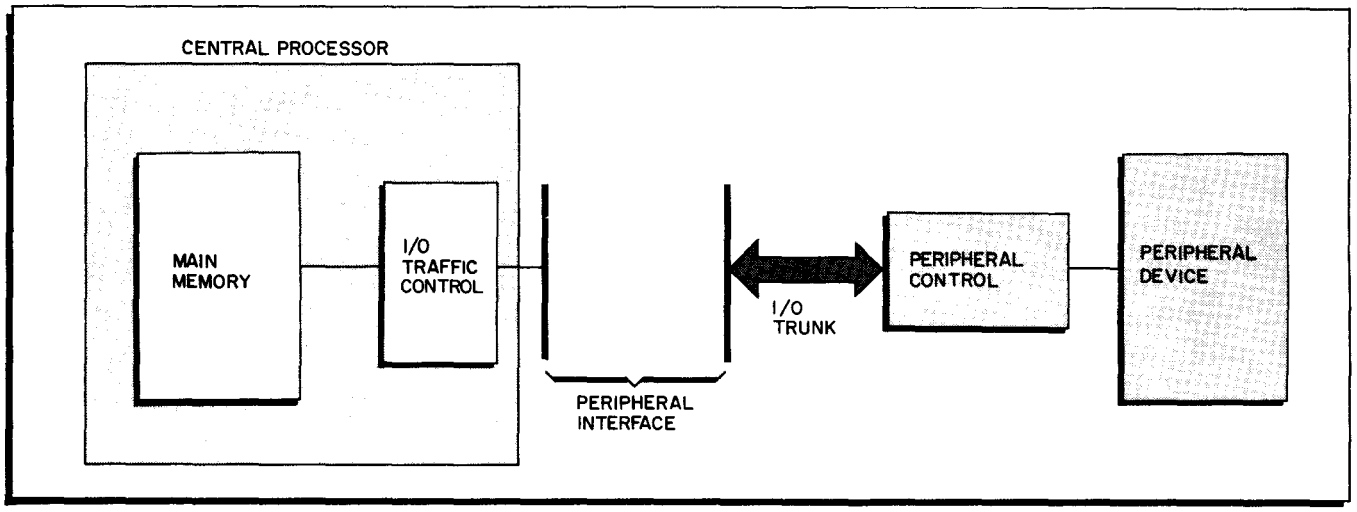


Figure 1-8. Basic Input/Output Data Path

Input/Output Trunk

An input/output trunk permanently connects a peripheral control (and its associated device) to the peripheral interface. The trunk transfers data either to or from the central processor and is therefore either an input or an output trunk, depending on the type of device it connects to the peripheral interface. For example, the I/O trunk connecting the card reader and its associated control is an input trunk, while the I/O trunk connecting the printer and its control is an output trunk. A peripheral control which performs both input and output functions (e.g., a magnetic tape control) requires two I/O trunks: one for input operations, and one for output operations.

The maximum number of peripheral controls that can be connected to a Series 200 processor is determined by the number of I/O trunks associated with that processor. For example, the Type 2201 can contain up to 32 I/O trunks, which means that as many as 32 peripheral controls can be attached to the processor at one time (see Figure 1-10).

Read/Write Channel

Notice that the data path shown in Figure 1-8 is incomplete: there is no connection across the peripheral interface. This final link in the data path, known as a "read/write channel," is inserted when the instruction is executed. Unlike an I/O trunk, which is permanently connected to a peripheral control, the read/write channel is assigned by the programmer to specialize the data path between a peripheral control and the processor.

When the programmer codes a Peripheral Data Transfer instruction, he specifies among other things the peripheral control that is to send or receive the data (and therefore the I/O trunk connected to that control) and the read/write channel 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 1-9 shows the data path formed during the execution of a Peripheral Data Transfer instruction in which the programmer specifies that the card reader control is to transfer data over read/write channel 2 (RWC2). 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.

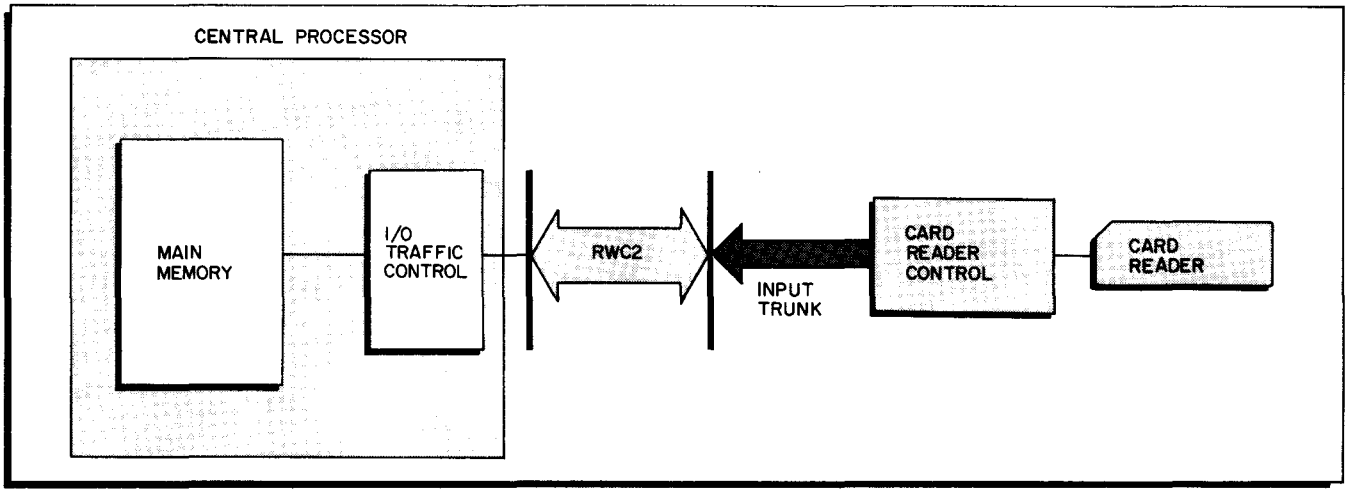


Figure 1-9. Data Path During Card Read Operation

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 indicates the number of peripheral operations that can be performed simultaneously by that processor (see Figure 1-10).

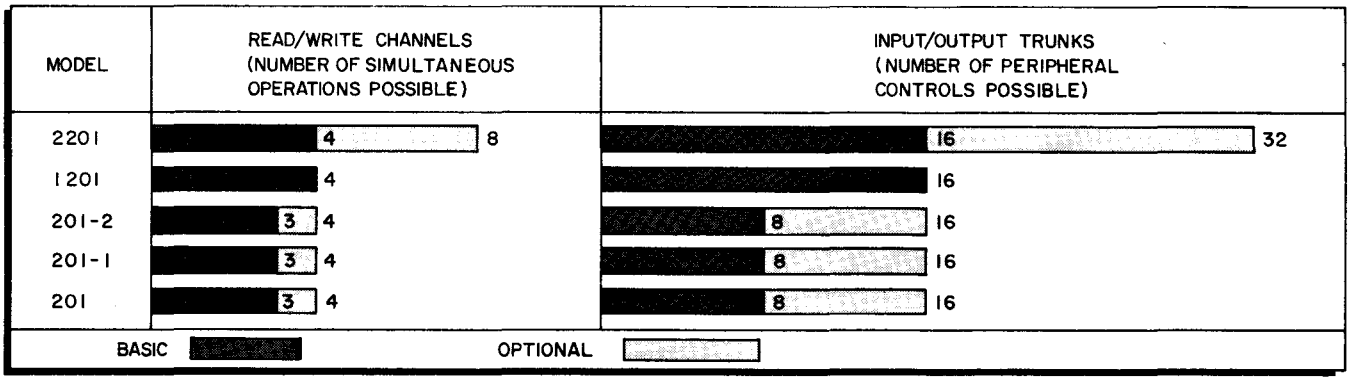


Figure 1-10. Data Path Components of Series 200 Processors

OPTIONAL FEATURES

Table 1-8 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 1-8. Series 200 Optional Features

	FEATURE	201	201-1	201-2	1201	2201
010	ADVANCED PROGRAMMING	n/a	n/a	OPT		
011	ADVANCED PROGRAMMING	OPT	OPT	n/a	n/a	n/a
012	PROGRAM INTERRUPT	OPT				
013	EDIT INSTRUCTION	OPT	OPT	OPT		
015	8 ADDITIONAL I/O TRUNKS	OPT	OPT	OPT		
016	AUXILIARY READ/WRITE CHANNEL	OPT	OPT	OPT		
1100	SCIENTIFIC UNIT	n/a	n/a	n/a	OPT	OPT
1115	16 ADDITIONAL I/O TRUNKS & 4 ADDITIONAL RWC'S	n/a	n/a	n/a	n/a	OPT
1114	STORAGE PROTECT	n/a	n/a	n/a	OPT	n/a
1117	STORAGE PROTECT	n/a	n/a	n/a	n/a	OPT

STANDARD OPT OPTIONAL

ADVANCED PROGRAMMING

Two Advanced Programming Features increase the basic instruction repertoire of the Model 200 processors. Feature 011 is available with the Types 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 1-9):

1. Additional program instructions.
2. The ability to modify instruction addresses via indexed or indirect addressing (described in Section 4).
3. A "read reverse" capability with magnetic tape units.

Table 1-9. 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. Extended Move 6. Move and Translate 7. Branch on Character Condition (expanded version) 8. Branch on Bit Equal[2]	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 2. Indirect addressing
Read Reverse	Read Reverse
Any Model 200 processor can read 1/2-inch magnetic tapes in a reverse direction and transfer the information to the main memory 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. It is included in the standard instruction repertoire of the Type 201-2 processor.

[2] The Branch on Bit Equal instruction is optionally available only with the Type 201-2 processor. The Load Control Registers instruction, optional with the Types 201 and 201-1 processors, is included in the standard instruction repertoire of the Type 201-2 processor.

[3] The Types 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 Types 1201 and 2201 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 8-102.

ADDITIONAL INPUT/OUTPUT TRUNKS AND READ/WRITE CHANNELS

Any information transferred between the central processor and a peripheral device is transmitted over a "data path" formed by a read/write channel and an input/output trunk. (The significance of these two elements is described on page 1-13.) The degree of peripheral simultaneity achievable by a processor and the number of peripheral devices connectable to that processor depends on the number of read/write channels and input/output trunks available, respectively. Three optional features allow a user to increase his processor's peripheral flexibility by adding the following elements:

1. Feature 015 — Eight additional input/output trunks for a Model 200 processor.
2. Feature 016 — One additional (auxiliary) read/write channel for a Model 200 processor.
3. Feature 1115 — Four additional read/write channels and 16 additional input/output trunks for the Type 2201 processor. The input/output trunks of Feature 1115 can be used only in conjunction with the read/write channels of this feature.

SCIENTIFIC UNIT

The scientific unit, which is physically contained in a separate unit of Series 200 cabinetry, adds 14 scientifically oriented instructions to the Series 200 repertoire. Available with the Types 1201 and 2201 processors, it is summarized in Appendix F and described in detail in the Honeywell Information Bulletin entitled Scientific Unit for Model 1200 and 2200 (Feature 1100).

STORAGE PROTECT

Two Storage Protect Features, identical in nature, are offered to the Type 1201 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. Any 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 Feature is described in Appendix E.

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.

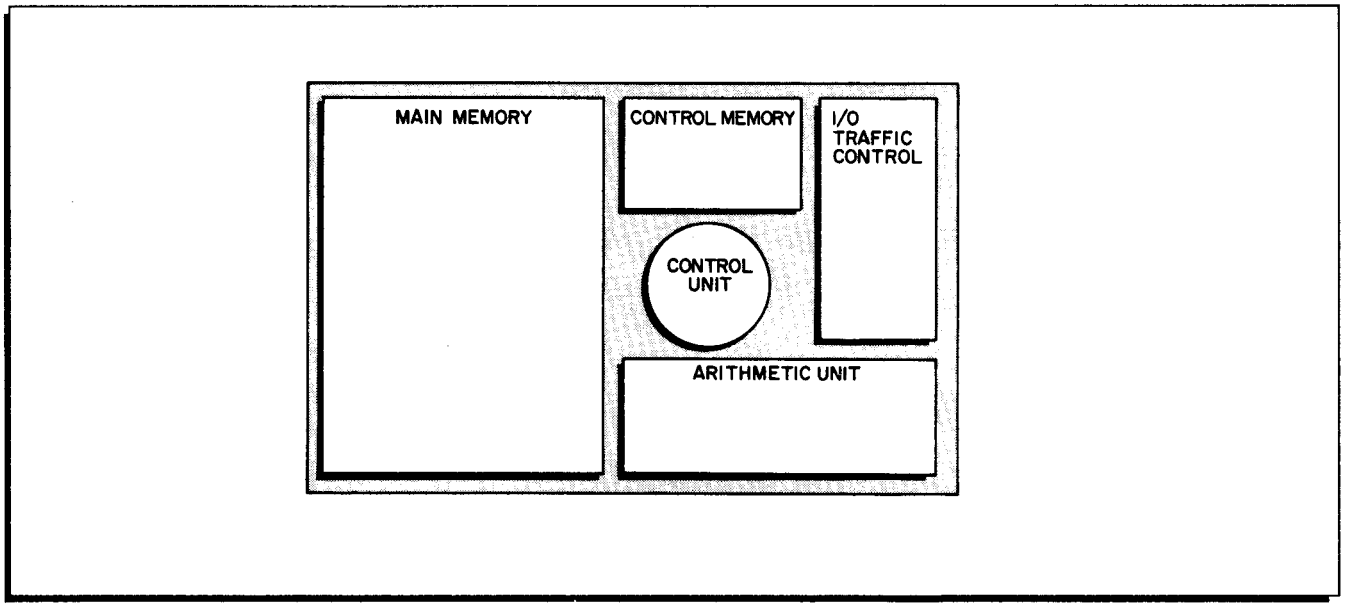


Figure 2-1. Logical Division of Series 200 Central Processor

MAIN MEMORY

The main memory contains from 2,048 to 262,144 character locations of magnetic core storage which are used to store program instructions and data during a program run (see Figure 2-2).

Nine planes of cores (see Figure 2-3) are placed on top of one another to form a memory "stack"; nine cores aligned vertically form a character position in memory. Every character position 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.

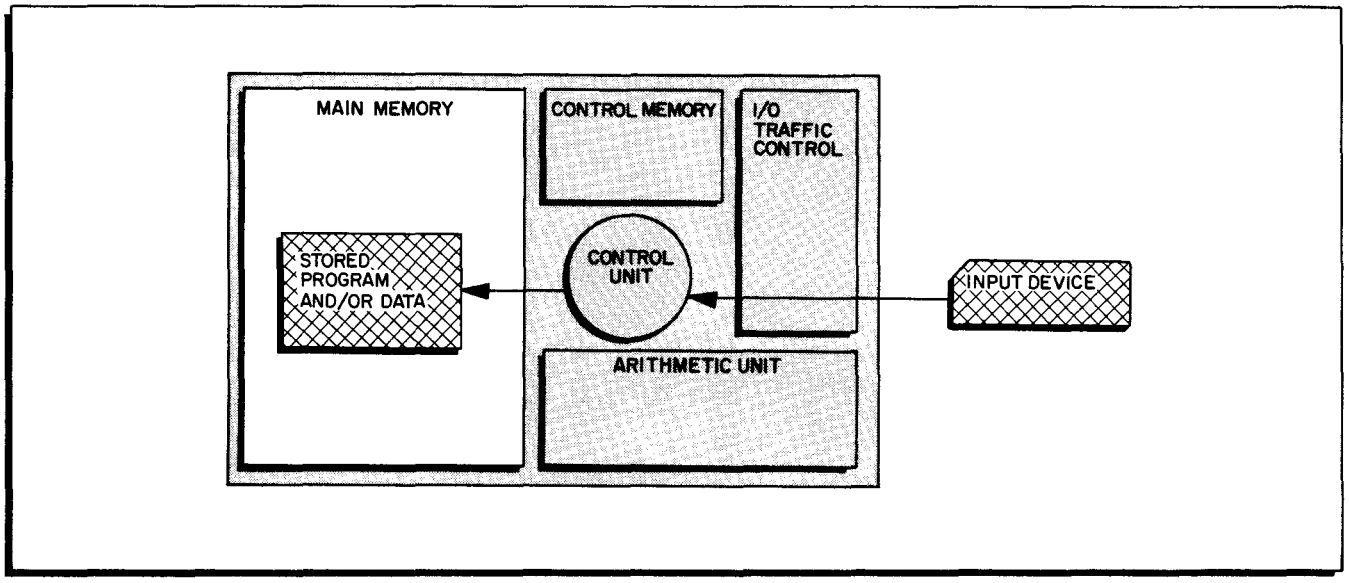


Figure 2-2. Main Memory Functions

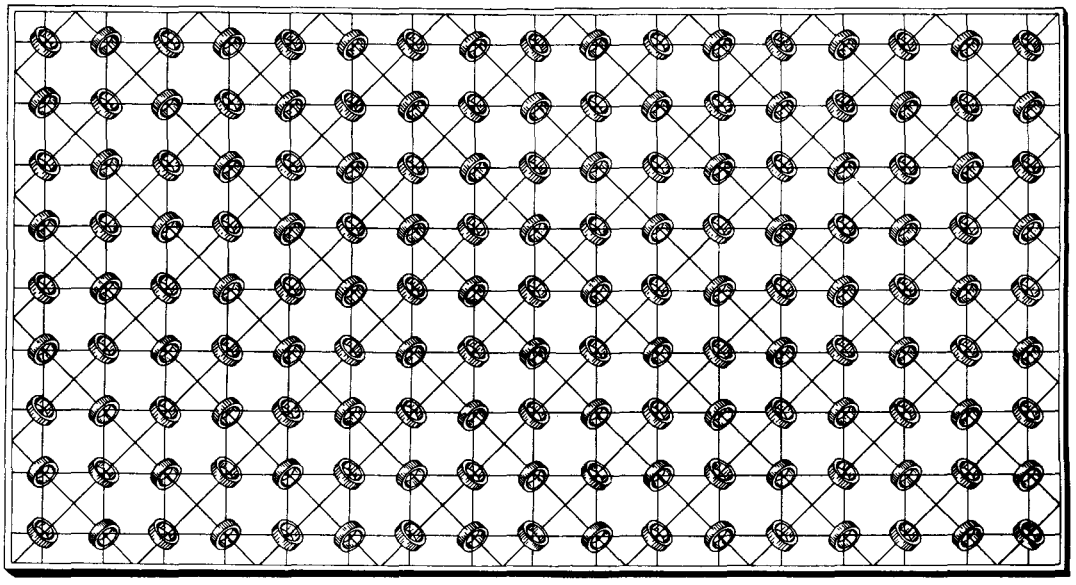


Figure 2-3. A Main Memory Core Plane

Figure 2-4 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 2-4, 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 2-5 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 numeric (1, 2, 4, and 8) and the A and B cores. The A and B cores correspond to card zone punches: the A bit represents a 12-punch, the B bit represents an 11-punch, a combination of the A and B bits represent a 0-punch. A listing of the main memory formats for all valid Series 200 characters appears in Appendix B.

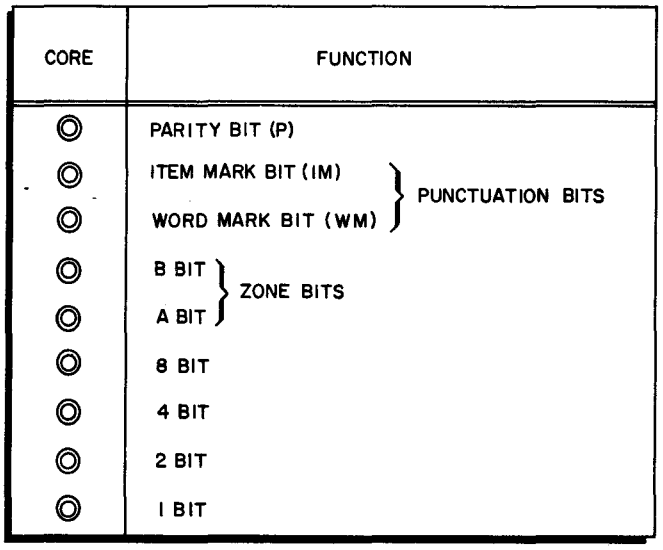


Figure 2-4. One Memory Position

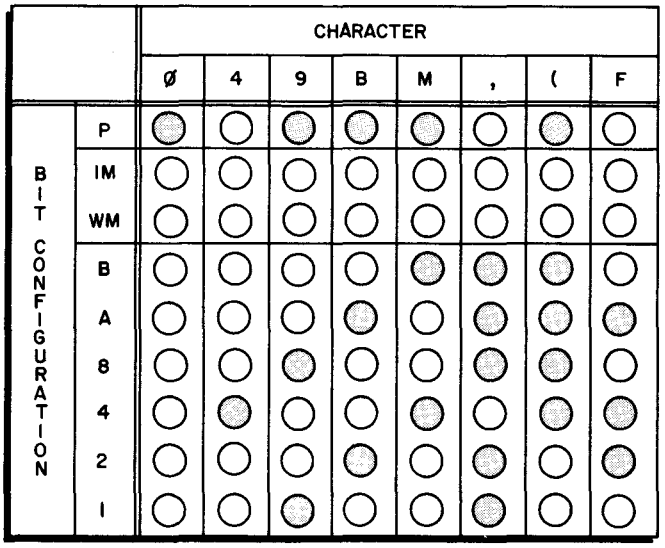


Figure 2-5. Representation of Characters in Magnetic Core Storage

The word-mark bit (WM) is used to define 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 3, the word-mark bit is instrumental in defining the size of such fields.

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 3).

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 3).

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 one-bits. Whenever a character is moved from one location to another it is automatically checked to determine if an odd number of bits has been moved. In Figure 2-5, the characters 0, 9, B, M, and (are represented by an even number of information bits. Circuitry within the central processor automatically adds a one in the parity bit positions of these characters to provide the required odd bit count.

CONTROL MEMORY

The control memory is a magnetic core storage unit consisting of up to 37 individually addressable control registers.[1] (The number of registers actually available depends on the system configuration.) Normally, control registers contain the addresses of instructions and of the data being processed during a program run. One such register, called the A-address register, is illustrated in Figure 2-6. 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.

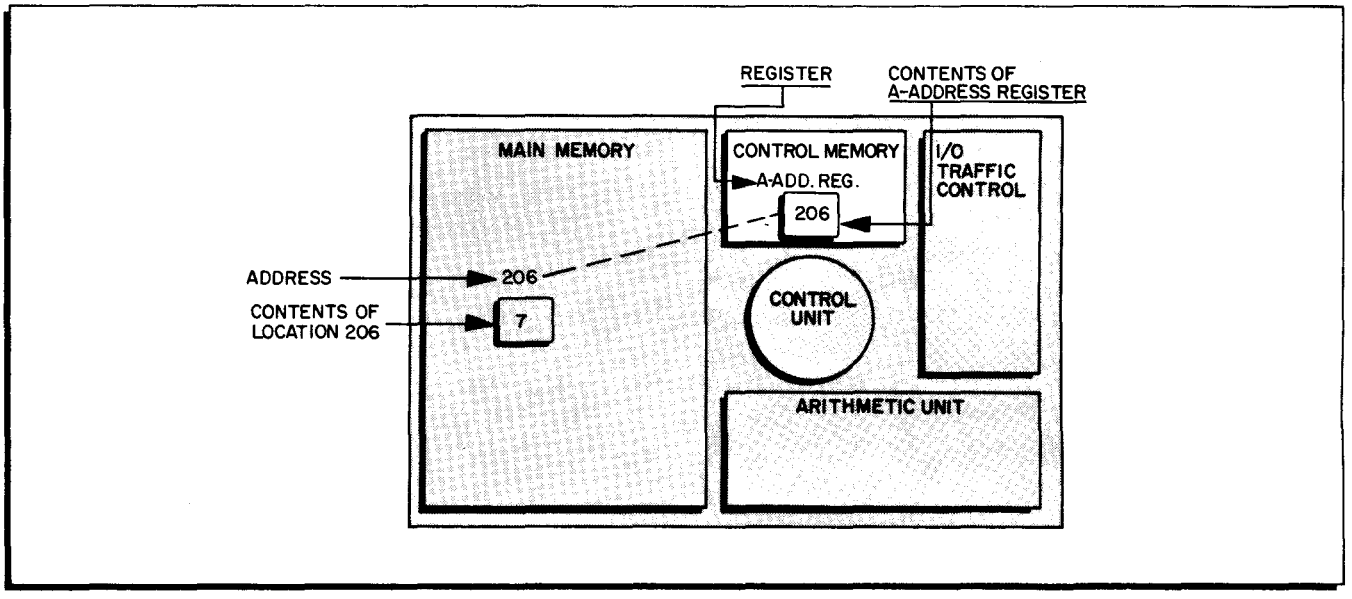


Figure 2-6. Typical Control Register Function

When the Scientific Unit (Feature 1100) is included in a Series 200 processor, each control register is three characters (18 bits) in length. When the Scientific Unit is not present, each control register is only as large as it need be to contain the largest (or "highest") main memory address in a user's processor. (The binary addressing technique used by Series 200 processors is described in Section 4.) Thus, a processor whose main memory capacity is 8,192 characters

[1] When the Series 200 model is equipped with the Scientific Unit (see Appendix F), 12 control memory locations form four floating-point accumulators; these registers should only be addressed by the scientific instructions included in that feature.

contains control memory registers which are each 13 bits long (13 bits allow 8,192 addresses), while the control registers of a processor containing 131,072 characters of main memory storage are each 17 bits long (see Table 2-1).

Table 2-1. Size of Control Memory Registers

MAIN MEMORY CAPACITY (Characters)	4,096	8,192	16,384	32,768	65,536	131,072	262,144
SIZE OF CONTROL MEMORY REGISTER (Bits)	12	13	14	15	16	17	18

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 2-2.

Table 2-2. Control Memory Registers

MNEMONIC DESIGNATION	FUNCTION	VARIANT CHARACTER
REGISTERS STANDARD IN ALL PROCESSORS		
1. AAR	A-Address Register	67
2. BAR	B-Address Register	70
3. SR	Sequence Register	77
4. CLC1	Read/Write Channel 1 – Current Location Counter	01
5. CLC2	Read/Write Channel 2 – Current Location Counter	02
6. CLC3	Read/Write Channel 3 – Current Location Counter	03
7. SLC1	Read/Write Channel 1 – Starting Location Counter	11
8. SLC2	Read/Write Channel 2 – Starting Location Counter	12
9. SLC3	Read/Write Channel 3 – Starting Location Counter	13
10. WR1	Work Register 1	75
11. WR2	Work Register 2[1]	74
12. WR3	Work Register 3	60
FEATURE 010 or 011		
13. CSR	Change Sequence Register	64

Table 2-2 (cont). Control Memory Registers

MNEMONIC DESIGNATION	FUNCTION	VARIANT CHARACTER
	FEATURE 012	
14. EIR	External Interrupt Register	66
	FEATURE 016	
15. CLC1'	Read/Write Channel 1' - Current Location Counter	05
16. SLC1'	Read/Write Channel 1' - Starting Location Counter	15
	FEATURE 1115	
17. CLC4	Read/Write Channel 4 - Current Location Counter	21
18. CLC5	Read/Write Channel 5 - Current Location Counter	22
19. CLC6	Read/Write Channel 6 - Current Location Counter	23
20. CLC4'	Read/Write Channel 4'- Current Location Counter	25
21. SLC4	Read/Write Channel 4 - Starting Location Counter	31
22. SLC5	Read/Write Channel 5 - Starting Location Counter	32
23. SLC6	Read/Write Channel 6 - Starting Location Counter	33
24. SLC4'	Read/Write Channel 4'- Starting Location Counter	35
	FEATURE 1100	
25.		41
26. AC0	Floating-Point Accumulator 0	42
27.		43
28.		45
29. AC1	Floating-Point Accumulator 1	46
30.		47
31.		51
32. AC2	Floating-Point Accumulator 2	52
33.		53
34.		55
35. AC3	Floating-Point Accumulator 3	56
36.		57
	FEATURE 1114 OR 1117	
37. IIR	Internal Interrupt Register	76

[1]Not accessible to the program.

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 4, "Addressing."

READ/WRITE COUNTERS

Data is transferred between the main memory and a peripheral device via a read/write channel (described in Section 1). 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, the address of the character position immediately following the last character transferred is stored in the current location counter.

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

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 capable of storing a single six-bit character. In general terms, an arithmetic or logic operation is performed as follows (see Figure 2-7):

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 operated upon.
2. The operands are transferred to the operand storage registers a character at a time, beginning with the rightmost character in each operand.
3. Each pair of characters that enters the storage registers is combined by the adder and the result is stored in the main memory as specified by the stored program instruction. If a carry is generated, it is stored in the adder and combined with the next higher-order pair of characters.

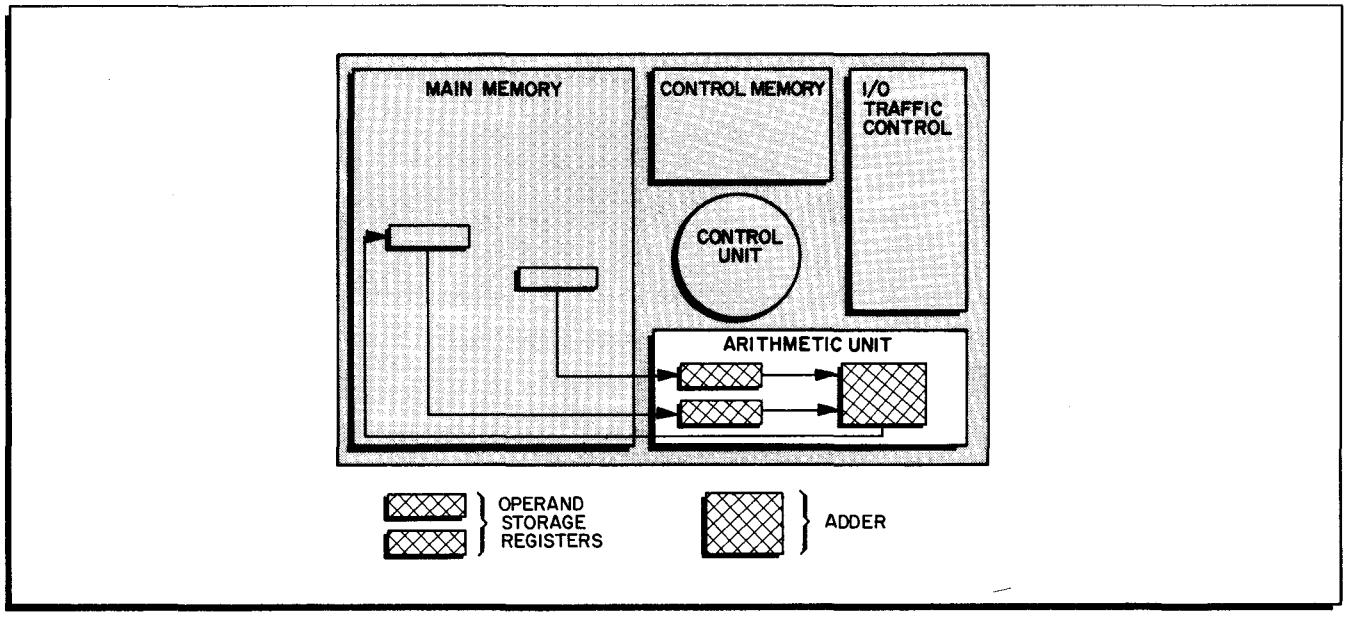


Figure 2-7. Data Flow Between Main Memory and Arithmetic Unit

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

CONTROL UNIT

The control unit is the hub of central processor activities (see Figure 2-8). 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.

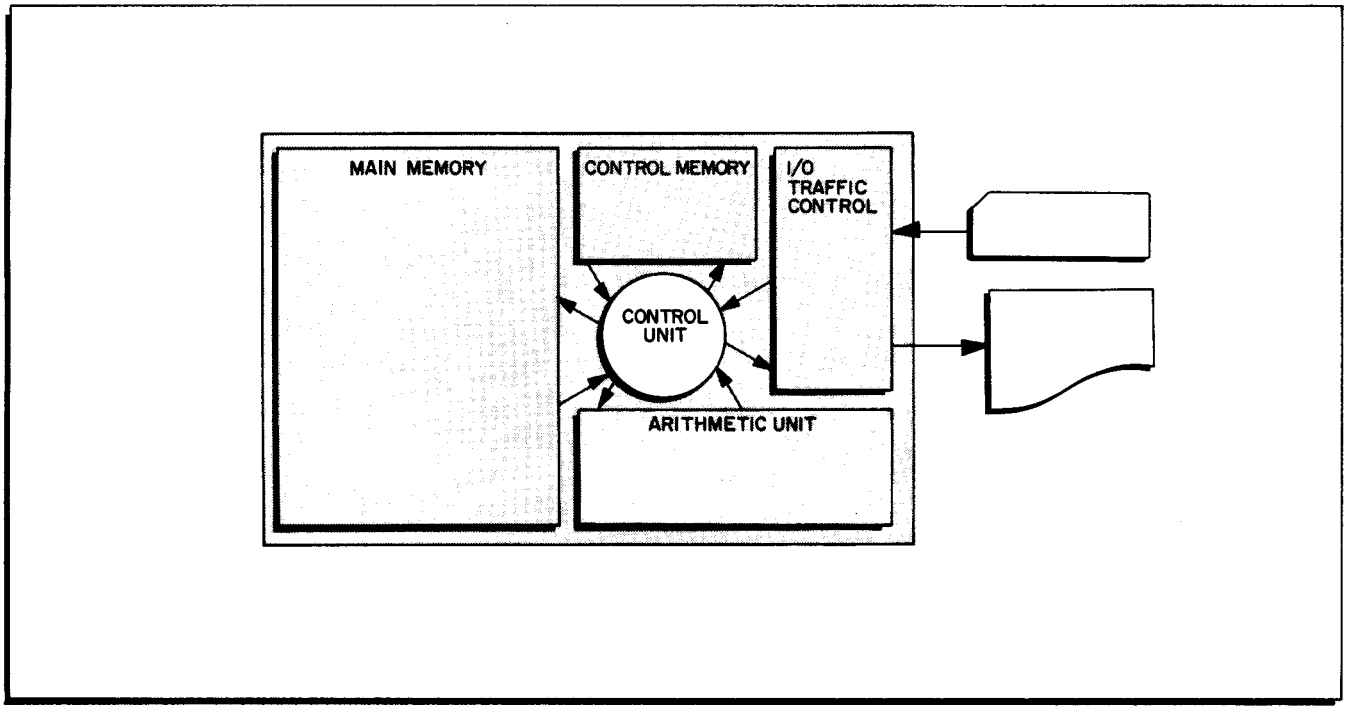


Figure 2-8. Control Unit Activities

INPUT/OUTPUT TRAFFIC CONTROL

The input/output traffic control is, as its name implies, the control 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 2-9).

The I/O traffic control enables from three (Model 200 minimum) to eight (Model 2200 maximum) simultaneous input/output operations to occur concurrently with the internal computations of the processor. This simultaneity is achieved by the traffic control's allocation of consecutive memory cycles to either peripheral controls or the central processor.

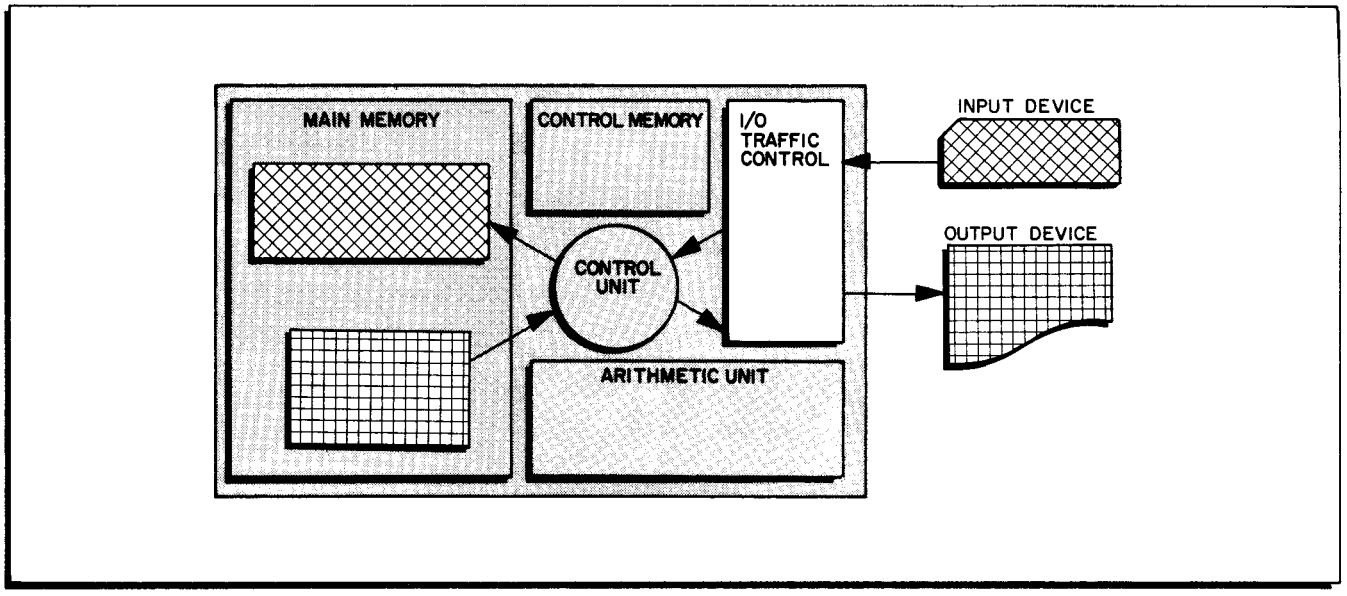


Figure 2-9. Input/Output Traffic Control Activities

MEMORY CYCLE DISTRIBUTION

When peripheral operations are in progress, a variety of mechanical activities may be taking place – paper advancing in a printer, a tape reel backspacing, a magnetic tape strip being selected, etc. During peripheral operations, only a fraction of actual central processor time is required to transfer information to and from the main memory; most of the time is taken up by the peripheral mechanical activities. The periods in which the central processor is actually interrupted for data transfer are spaced over the duration of the peripheral operation (see Figure 2-10).

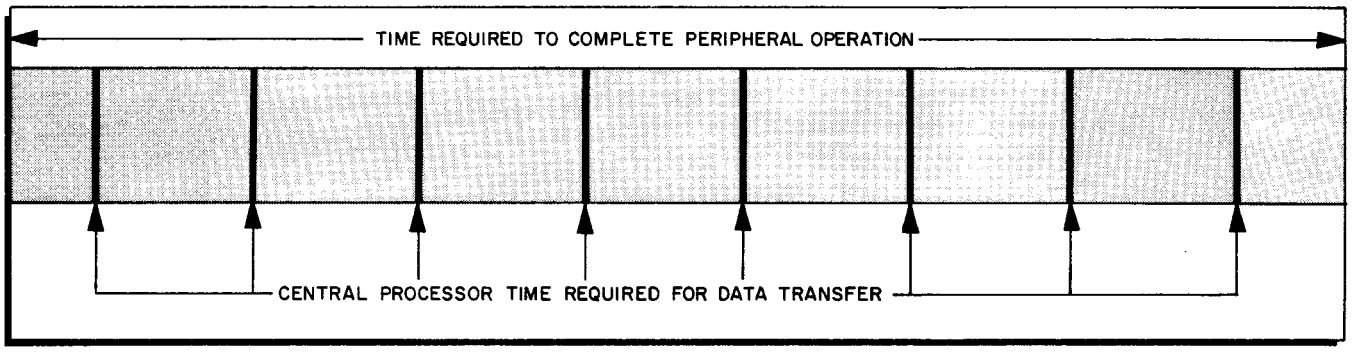


Figure 2-10. Data Transfer Intervals During One Peripheral Operation

When a peripheral operation is in progress but is not using main memory (the gray areas in Figure 2-10), 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.

The rate at which each peripheral control transfers data over a programmer-assigned read/write channel depends on the mechanical characteristics of the device connected to the control.[1] Thus, the transfer intervals shown in Figure 2-10 are spaced according to the device being used. For instance, the transfer rate for the mass memory file is considerably faster than that for the card punch; therefore, the mass memory file will require access to the main memory more frequently than the card punch. The I/O traffic control monitors the requests for access to the main memory and insures that all requests are honored within the prescribed time interval for each unit. The manner in which this is done is illustrated in Figure 2-11. Essentially, the traffic control decides how each memory cycle should be used – by a read/write channel or by the central processor – as described below.

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. If the channel does not require the memory cycle (i.e., if there is no information to be transferred through the channel at that time), the memory cycle is given to the central processor for internal data processing.

Each basic read/write channel associated with a processor is granted a memory cycle access to the memory every six microseconds. Thus, the Model 200 processors grant a two-microsecond access to each one of the three basic read/write channels every six microseconds; the Type 2201 processor gives a one-microsecond memory access to each one of six basic channels every six microseconds. The Type 1201 processor also offers a memory cycle (1.5 microseconds) to each of the three basic channels every six microseconds but in a slightly different manner. There are four 1.5-microsecond memory cycles in every six-microsecond interval. Thus, RWC1, RWC2, and RWC3 are each granted a 1.5-microsecond access to the memory, and 1.5 microseconds is still available before the next six-microsecond interval begins. This "residual" memory cycle is always given to the Type 1201 processor for internal computation.

AUXILIARY READ/WRITE CHANNELS

RWC1' and RWC4' are called auxiliary read/write channels because of the manner in which they are granted access to the main memory by the input/output traffic control. RWC1 and

[1] Read/write channels are described in Section 1.

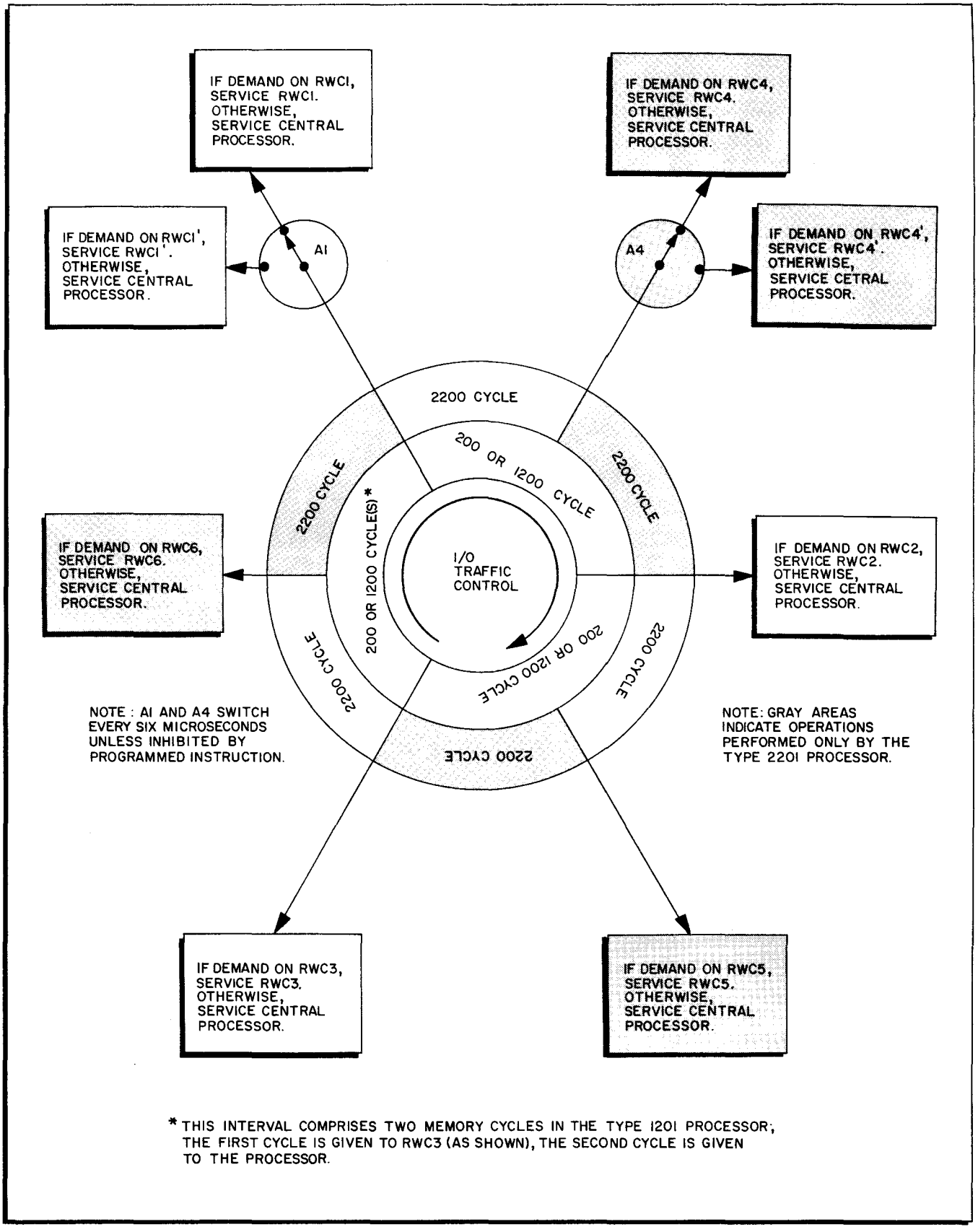


Figure 2-11. Symbolic Representation of Input/Output Traffic Control

RWC1' are connected to an alternator; RWC4 and RWC4' are also connected to an alternator. Every six microseconds, either or both alternators (depending on the number of channels associated with a processor) switch to allow one of the attached read/write channels access to the main memory. By providing alternate access between RWC's 1 and 1' and between RWC's 4 and 4', each auxiliary RWC can gain access to the main memory once every 12 microseconds.

It should be noted that program control of the alternators is required because the data transfer characteristics of certain peripheral devices require that main memory be accessed at intervals less than 12 microseconds. This means that when such devices are linked to the main memory via RWC1 and/or RWC4, the action of the appropriate alternator must be inhibited. Under no conditions should such devices be assigned to transmit data over RWC1' or RWC4'. As discussed in Section 8, input/output instructions can be specified to inhibit the action of the alternator(s), thereby guaranteeing RWC1 and/or RWC4 access to the main memory every six microseconds by denying memory access to the corresponding RWC(s).

Table 2-3. Summary of Central Processor Characteristics

PROCESSOR		201	201-1	201-2	1201	2201
MAIN MEMORY						
PROCESSING UNIT		Six-bit character. Groups of consecutive characters form instructions and data fields. Fields are defined by word mark punctuation (see Section 3).				
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	16,384-262,144
MEMORY CYCLE (Microseconds)		2	2	2	1.5	1
INDEX REGISTERS		0-6	0-15	0-15	15-30	15-30
CONTROL MEMORY						
MEMORY CAPACITY (Control Registers)		12-16	13-16	13-16	16-29	16-37
MEMORY CYCLE		0.50 microseconds				
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	36μs	24μs
	5-Digit Compare (A:B)	38μs	38μs	38μs	30μs	21μs

Table 2-3 (cont). Summary of Central Processor Characteristics

PROCESSOR	201	201-1	201-2	1201	2201
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	4-8
INPUT/OUTPUT TRUNKS	8-16	8-16	8-16	16	16-32
SIMULTANEOUS OPERATIONS POSSIBLE	3-4	3-4	3-4	4	4-8

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 models 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 length 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 will 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.

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 5, the task of coding the instructions in a program is greatly simplified by 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 5).

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.

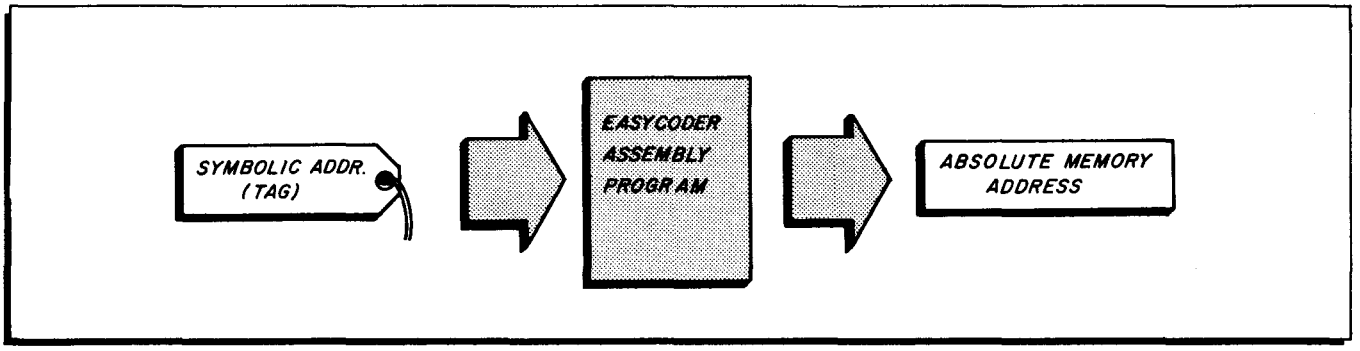


Figure 3-1. Conversion of Symbolic Tags to Absolute Memory Addresses

Because of the modular design of Series 200 models, 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 4).

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.

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 operations. The format of an input/output instruction (shown in Section 8 under the heading "Input/Output Instructions") is a modification of format 3 shown below. 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.

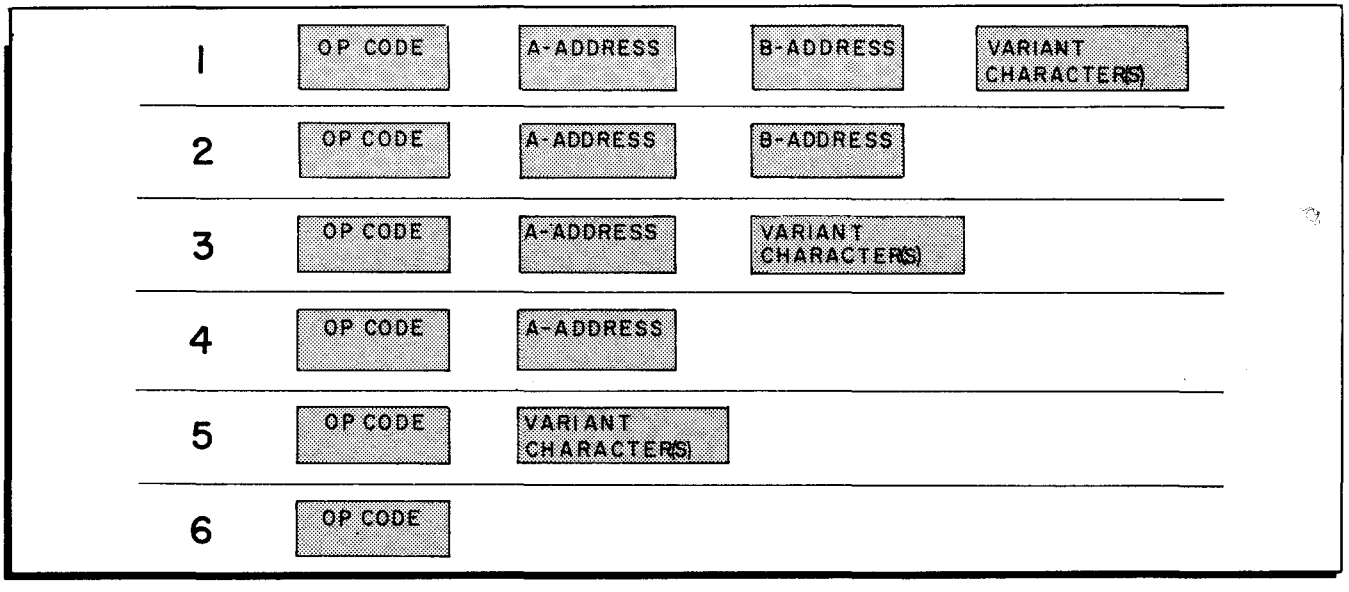


Figure 3-2. Series 200 Instruction Formats

For the sake of direct comparison, Figure 3-3 illustrates each of the formats defined in Figure 3-2 as a symbolic entry on the programmer's coding form.

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, 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 defined by setting a word mark in position 990. The right boundary is defined by specifying storage address 997 in the instruction that will manipulate the field. The eight-character group shown in Figure 3-5 is properly 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

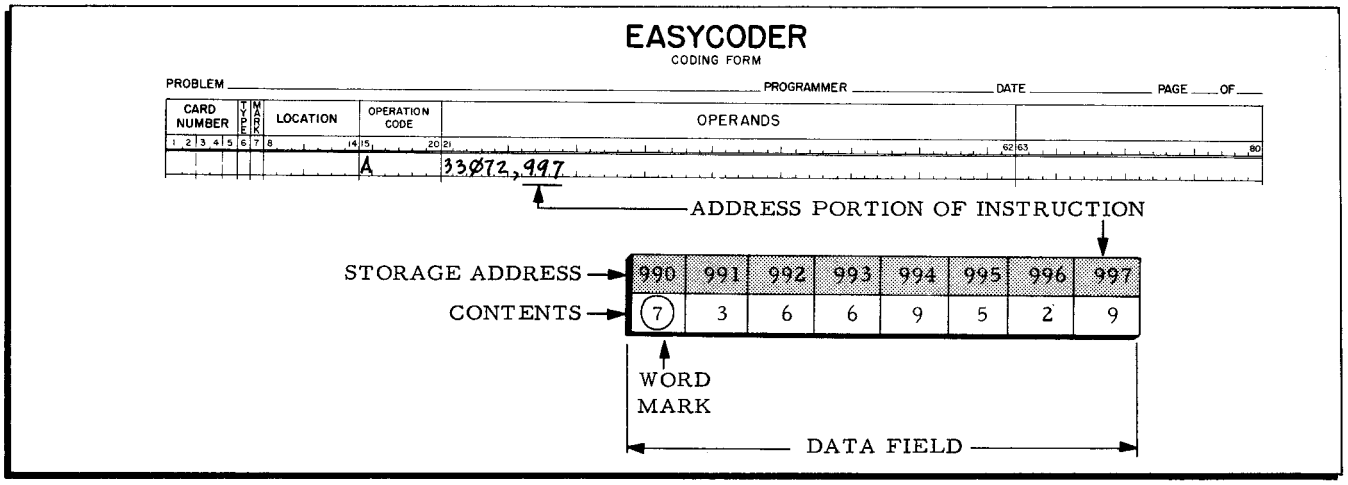


Figure 3-5. Data Field Format in Main Memory

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.

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

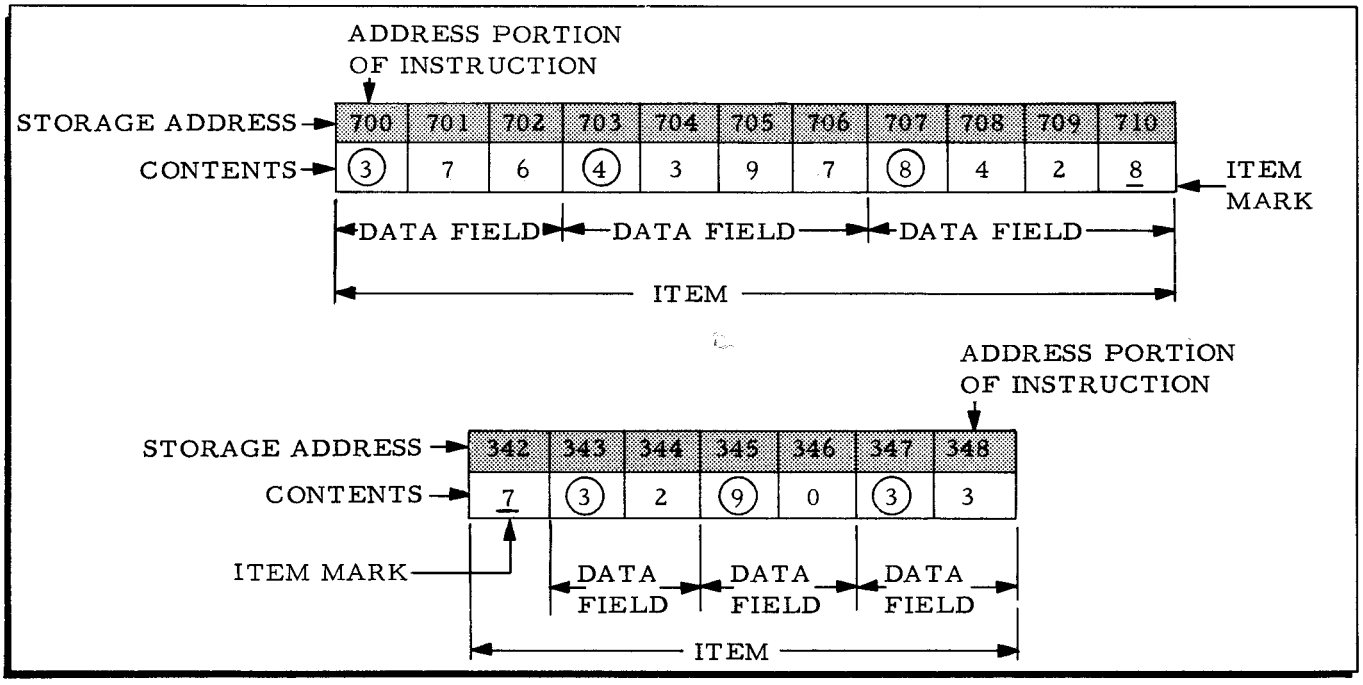


Figure 3-6. Two Item Formats in Main Memory

RECORDS

A record is any unit of information that is to be transferred between the main memory and a peripheral device.[1] 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).

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.

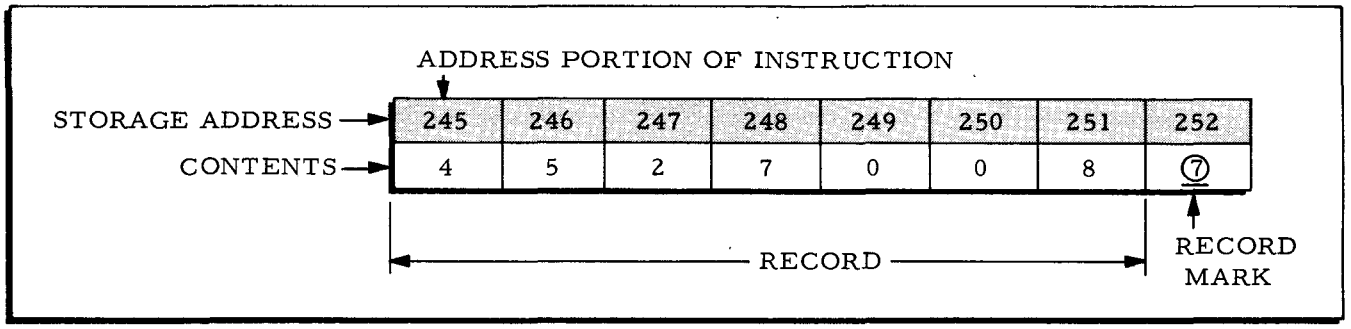


Figure 3-7. Record Format in Main Memory

SUMMARY

The foregoing data formats 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 X	Set Item Mark
	Item Mark X	Address portion of instruction	
RECORD	Address portion of instruction	Record mark ⊗ (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 8). In this case, the character containing the record mark is considered as part of the record.

MAGNETIC TAPE DATA FORMAT

In many applications, a major input and output medium for a Series 200 model 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 group 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.

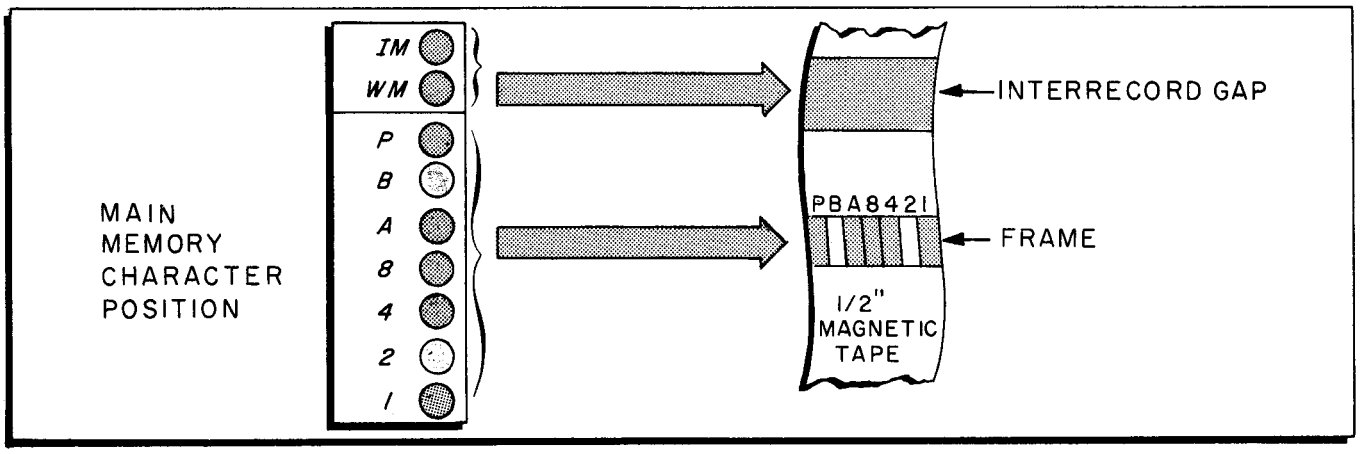


Figure 3-9. Character Representation on Magnetic Tape

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.

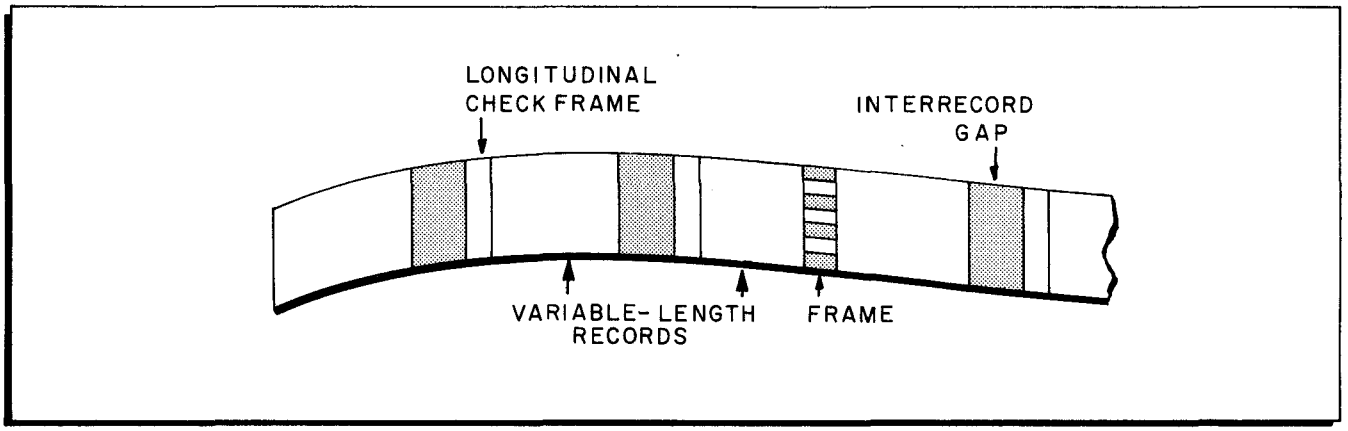


Figure 3-10. Data Format on Magnetic Tape

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).

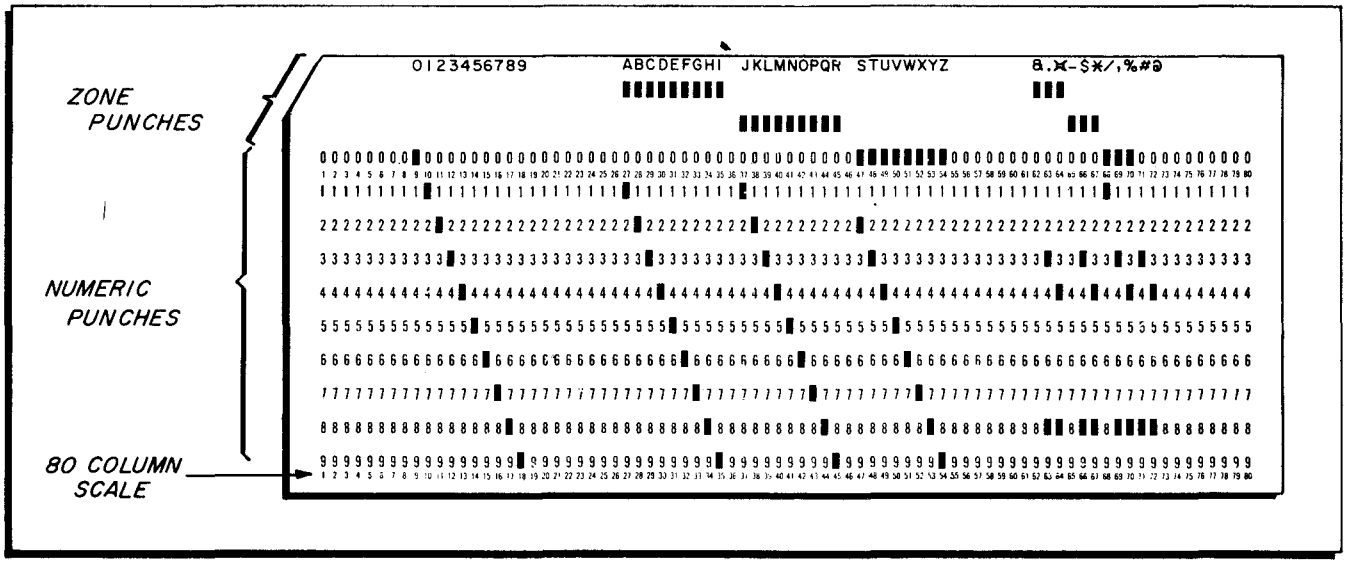


Figure 3-11. Punched Card Codes

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 tape strips, 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 one 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 Add instruction appears when stored in the main memory. (Recall that a character enclosed 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 storage location 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.

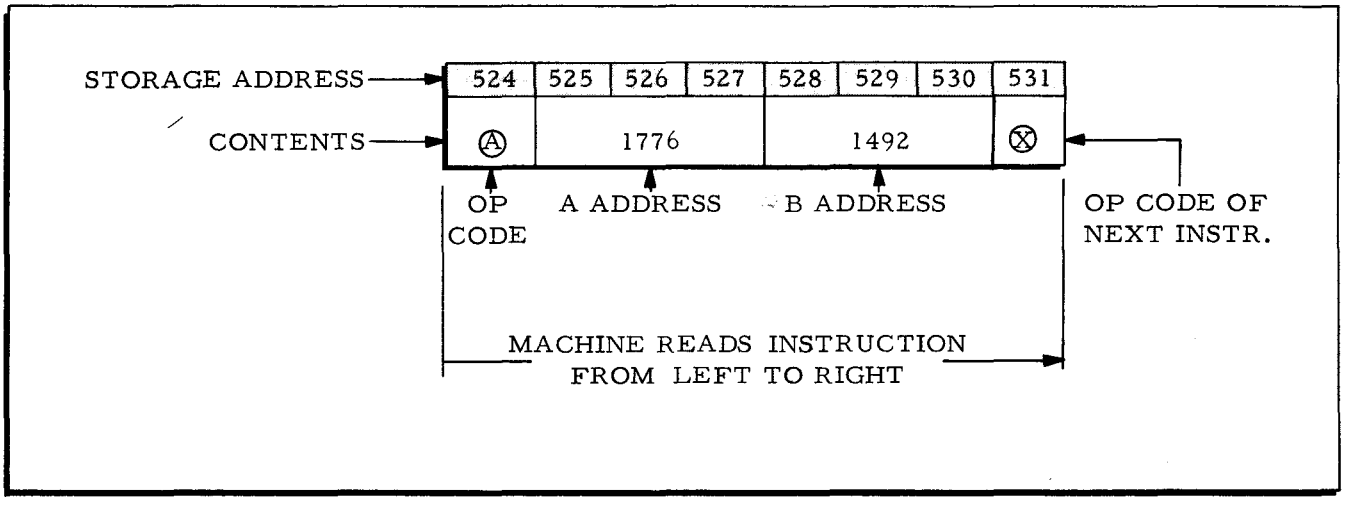


Figure 4-1. Typical Add Instruction

As mentioned in Section 3, a data field is 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. 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.[1]

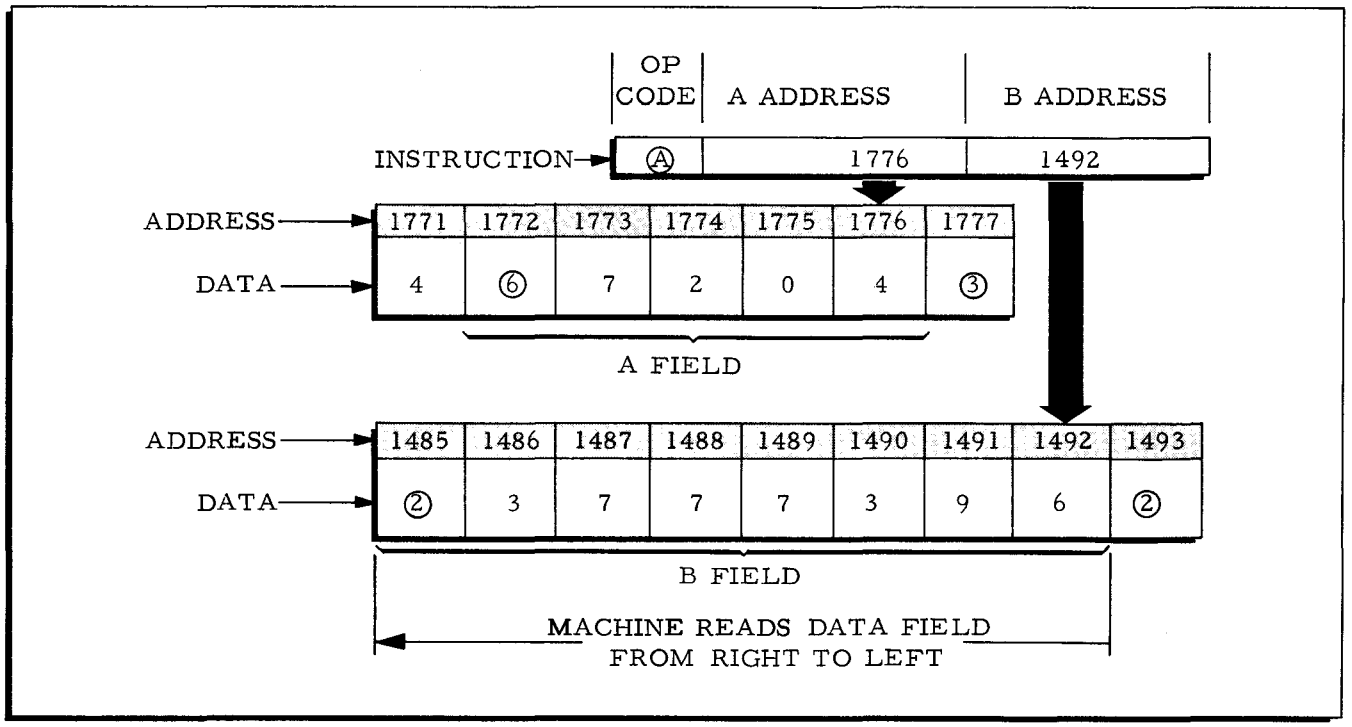


Figure 4-2. Extraction of Data Fields in Typical Add Instruction

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. Note that the contents of the character position containing a record mark are not considered as part of the record.

[1]NOTE: 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 is read from right to left becuase arithmetic 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) 17

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 8-72) 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 (see footnote below. 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

[1] A Load Control Registers instruction can be used to store the desired op code address (see page 8-67).

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 8-97) can be issued.

INTERNAL INTERRUPT REGISTER (IIR)

The address of an op code can also be stored in IIR.[1] When the Type 1201 or 2201 processor is equipped with the Storage Protect Feature, certain operations are considered as "violations" of storage protection (e.g., the attempt to transfer data from a peripheral control to 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) 14

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

B-ADDRESS REGISTER (BAR) 10

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 executed. If the B operand is a record or a leftmost-addressed item, the contents of BAR are incremented by one as each character is executed.

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. 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.[2] 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 8-67).

[2] A record also can be moved internally from right to left by means of the Extended Move instruction (see Section 8).

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 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 2-1), so that the size of the user's control registers ranges from 12 to 18 bits in length. The programmer should keep this fact in mind while reading the following description of addressing modes.

ADDRESSING MODES

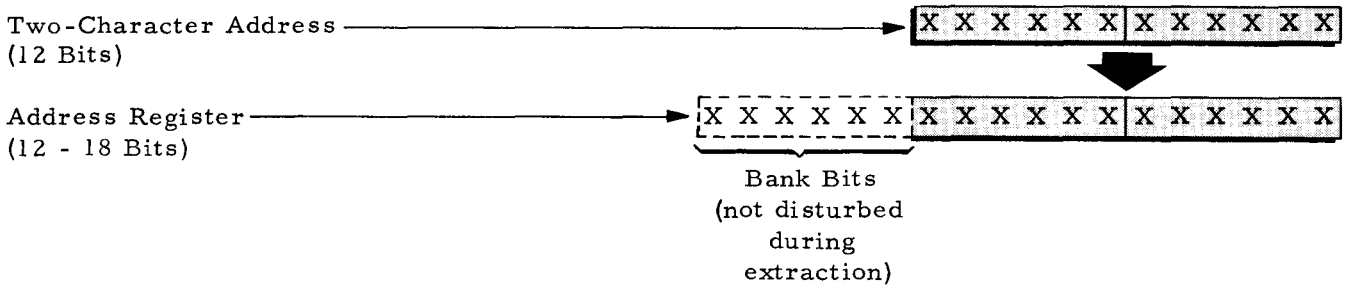
As stated at the beginning of this section, an instruction is stored in a field of from 1 to 12 characters, depending on the instruction's format and the programmed addressing mode. The op code is stored as a single, six-bit character. Variant characters or I/O control characters, if any, are each stored 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 8-69), by 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 4-16). 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 are 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", forming 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.

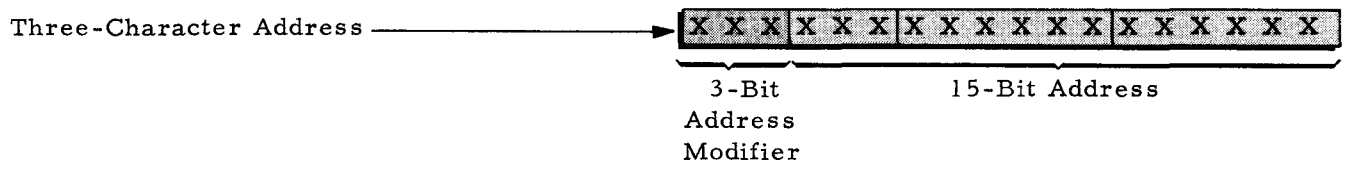
Indexed and indirect addressing (see below) cannot be performed in the two-character addressing mode.

THREE-CHARACTER ADDRESSING MODE

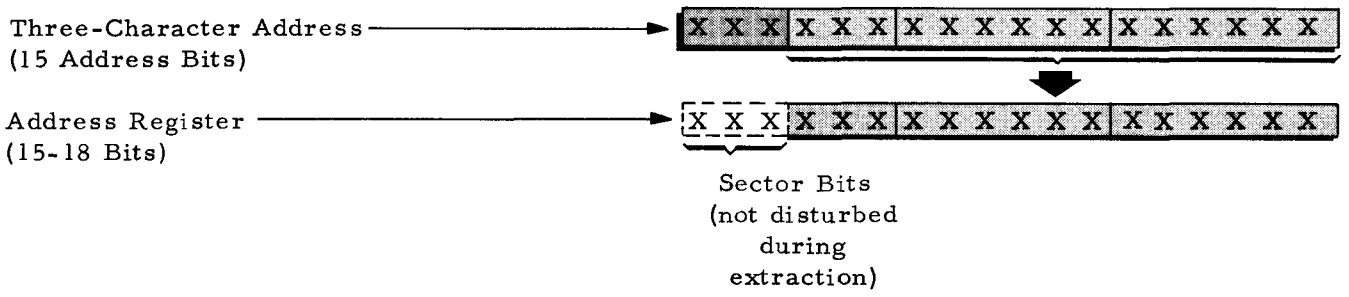
An operand address written in the three-character addressing mode is stored in three consecutive character locations of the memory. The rightmost 15 bits of the stored address represent the address of a main memory location in the range 0 - $32,767_{10}$. The leftmost three

[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]



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 are 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", forming 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

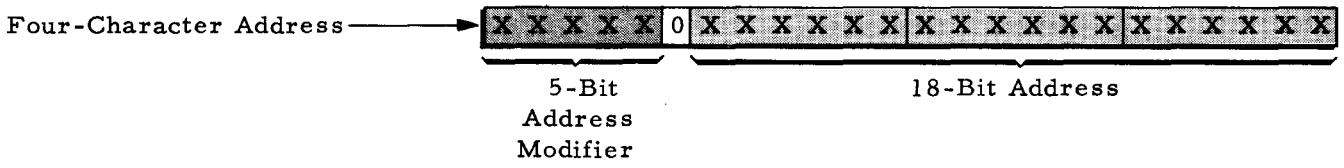
[1] The entire contents of an address register (sector bits + 15-bit address) are affected during the extraction of an instruction whose extraction path "duplicates A" (described on page 4-16). Extraction of all other three-character addresses affects only the rightmost 15 bits in the register.

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

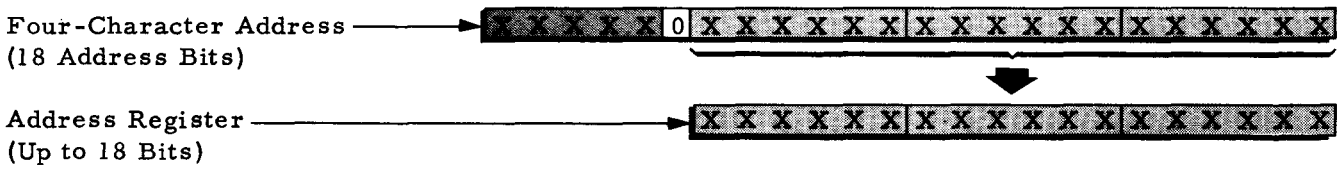
Addressing is continuous throughout the entire memory when a peripheral transfer operation is performed, as in the two-character mode.

FOUR-CHARACTER ADDRESSING MODE

An operand address written in the four-character addressing mode is stored in four consecutive character locations. The rightmost 18 bits represent a main memory address in the range 0 - $262,144_{10}$.[1] The leftmost five bits — the "address modifier" — specify whether the address is direct, indirect, or indexed (see "Address Modification," below).



The 18-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.



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 262,144 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 Series 200 processor containing the Advanced Programming Feature (Feature 010 or 011).

[1] The nineteenth bit of a four-character address is reserved for possible memory expansion. This bit is always zero in Series 200 processors with a main memory capacity of 262,144 characters or less.

These addressing forms are represented by the configuration of the "address modifier" as described below and are interpreted by the processor during the extraction phase.

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 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 4-3 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.

Indexed Addressing

A Series 200 processor can contain up to 30 index registers, depending on the type of processor and the optional features included in that processor. Table 4-1 summarizes the manner in which a processor acquires index registers.

Indexing operations in the three-character addressing mode use index registers 1 through 6. These registers are located in the first 25 locations (locations 0 through 24) of the 32,768-character sector in which the instruction is stored (see Table 4-2).[1]

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

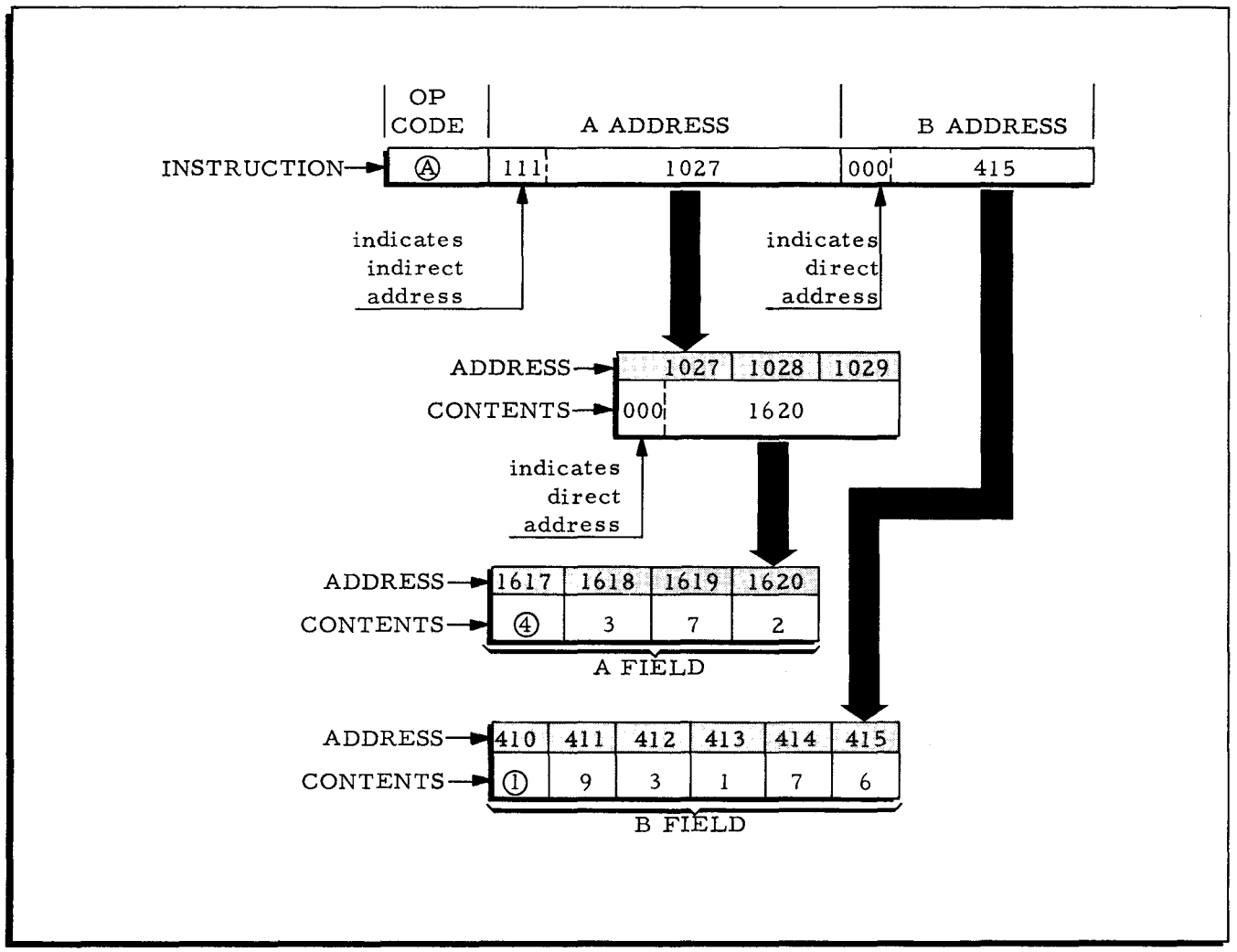


Figure 4-3. Extraction of Three-Character Indirect Address

Table 4-1. Number of Index Registers Available to Series 200 Processors

Type	Basic Processor	Features		Minimum	Maximum
		Advanced Programming (Feature 010 or 011)	Storage Protect (Feature 1114 or 1117)		
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
2201	15	n/a[1]	15	15	30

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

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

Index Register	Address Modifier	Storage Field	Address
1	001	2 - 4 (+n)	4 (+n)
2	010	6 - 8 (+n)	8 (+n)
3	011	10 - 12 (+n)	12 (+n)
4	100	14 - 16 (+n)	16 (+n)
5	101	18 - 20 (+n)	20 (+n)
6	110	22 - 24 (+n)	24 (+n)

n = first location of the 32,768-character sector in which the instruction is stored.

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. 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.

The modification of an address occurs in its respective address register. For instance if the B-address portion of an instruction is indexed, the indexing 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 4-4 illustrates how the Add instruction on page 4-10 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 5.

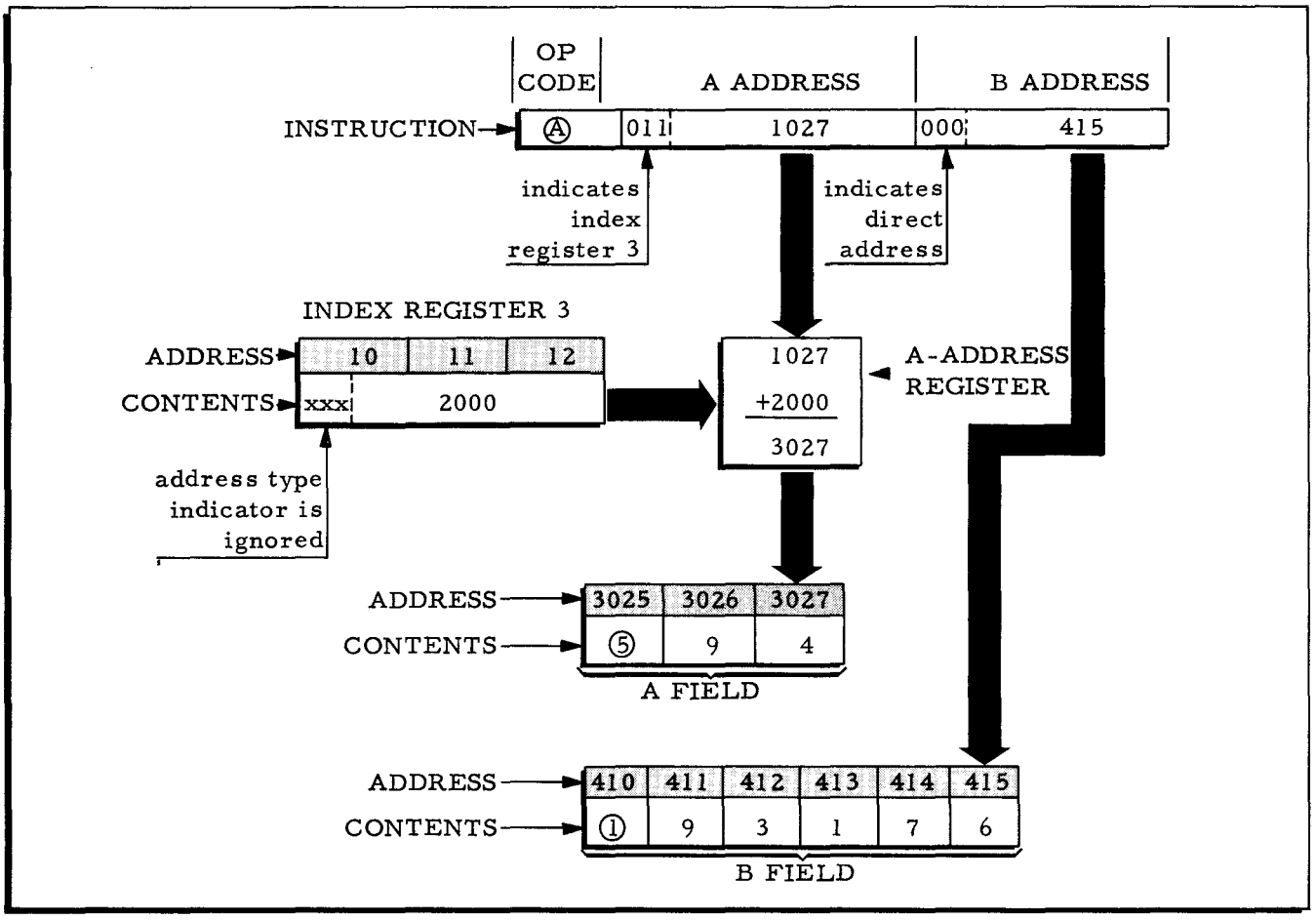


Figure 4-4. Extraction of Indexed Address in Three-Character Mode

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 5).

Indexed Addressing

Four-character indexed addresses to be modified by index registers 1 through 15 are specified by an address modifier whose configuration is 00001 through 01111, respectively. Index registers 16 through 30, when present,[1] are specified by the configurations 10001 through 11111 (see Table 4-3).

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

Index Register	Address Modifier	Storage Field	Address
1	00001	2-4	4
2	00010	6-8	8
3	00011	10-12	12
4	00100	14-16	16
5	00101	18-20	20
6	00110	22-24	24
7	00111	26-28	28
8	01000	30-32	32
9	01001	34-36	36
10	01010	38-40	40
11	01011	42-44	44
12	01100	46-48	48
13	01101	50-52	52
14	01110	54-56	56
15	01111	58-60	60
16	10001	Same as above, only relative to the 4,096-character memory bank designated by the Load Index/Barricade Register instruction (see page 8-84).	
17	10010		
18	10011		
19	10100		
20	10101		
21	10110		
22	10111		
23	11000		
24	11001		
25	11010		
26	11011		
27	11100		
28	11101		
29	11110		
30	11111		

Index registers 1 through 15 are located in the first 61 locations of the main memory (locations 0 - 60_{10}), each register occupying three storage locations. The situation of these registers is independent of the location of the instruction whose address(es) is indexed. Index registers 16 through 30 are located in the first 61 locations of the "protected" memory area in the Type 1201 or 2201 processor. (Recall that the main memory of the Type 1201 or 2201 can be logically divided

[1] Index registers 16 through 30 are the registers included in the Storage Protect Feature.

at any 4,096-character bank into an "open" area and a "protected" area. The specific bank at which the division takes place is specified by the Load Index/Barricade Register instruction described in Section 8.)

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).

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

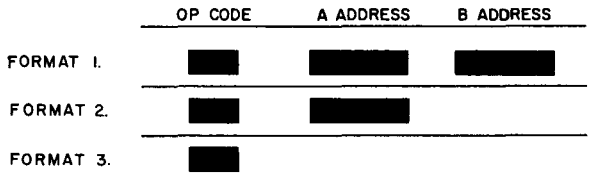
Main Memory Capacity (Chars.)	32,768	65,536	131,072	262,144
Number of Active Address Bits	15	16	17	18

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 4-5. Indirect addressing is specified in the A address, and indexed addressing (via index register 13) is specified in the B address.

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 4-6).

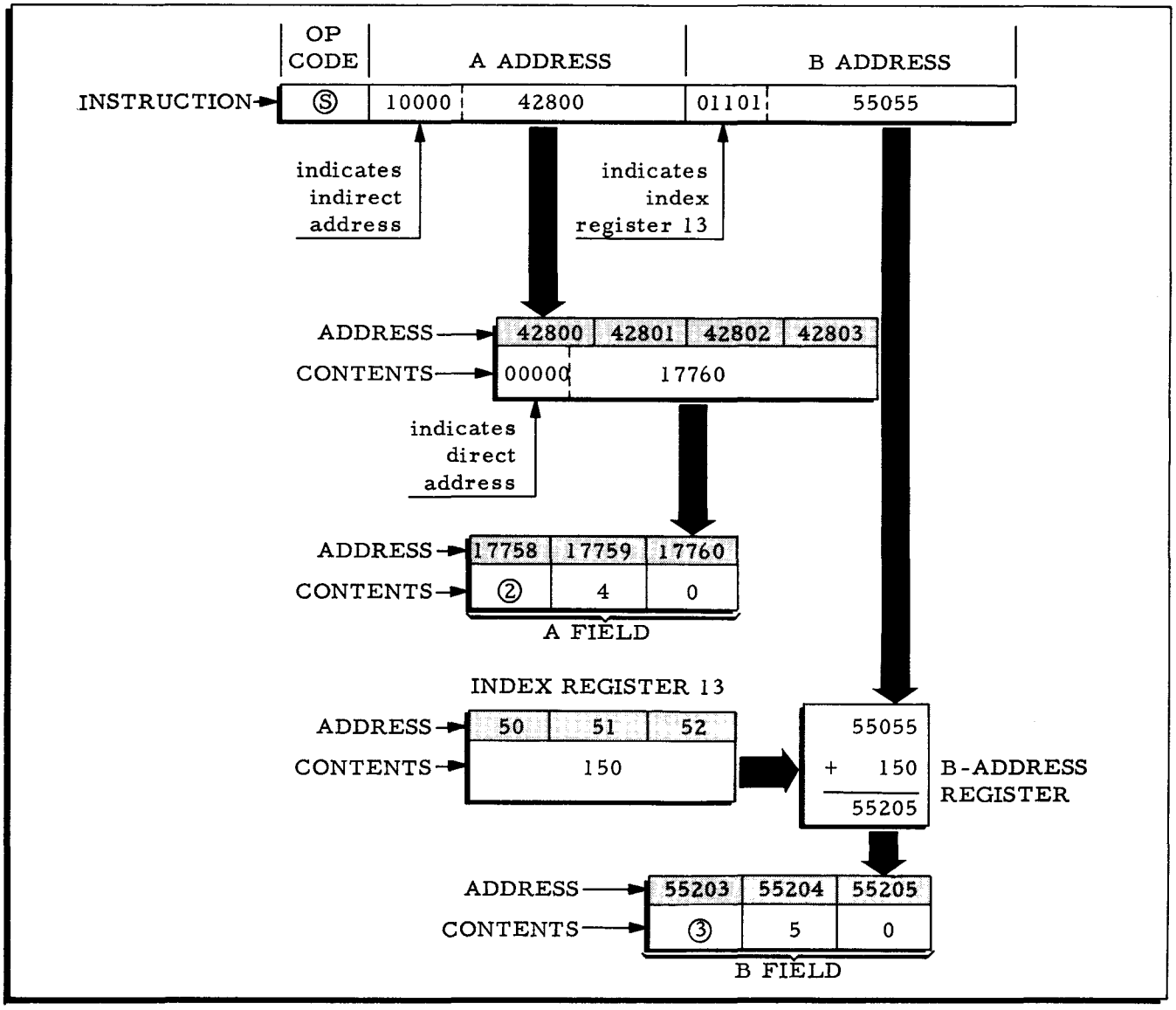


Figure 4-5. Extraction of Indirect and Indexed Four-Character Addresses

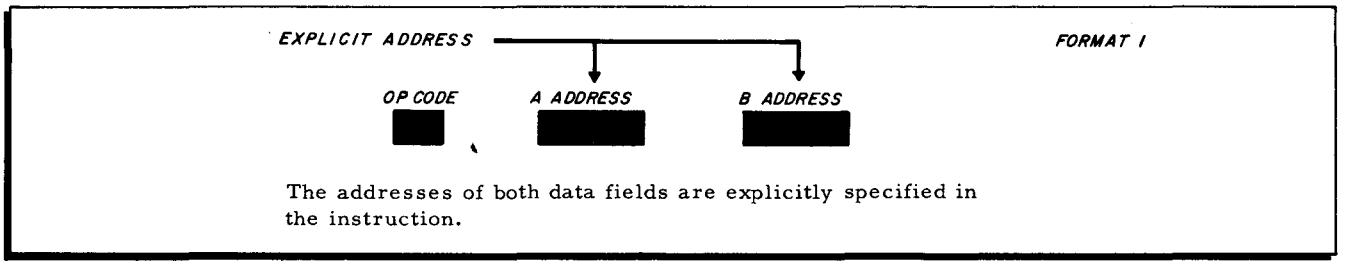


Figure 4-6. Series 200 Instruction Format 1

Format 2 has two possible interpretations (see Figure 4-7):

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 18 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).

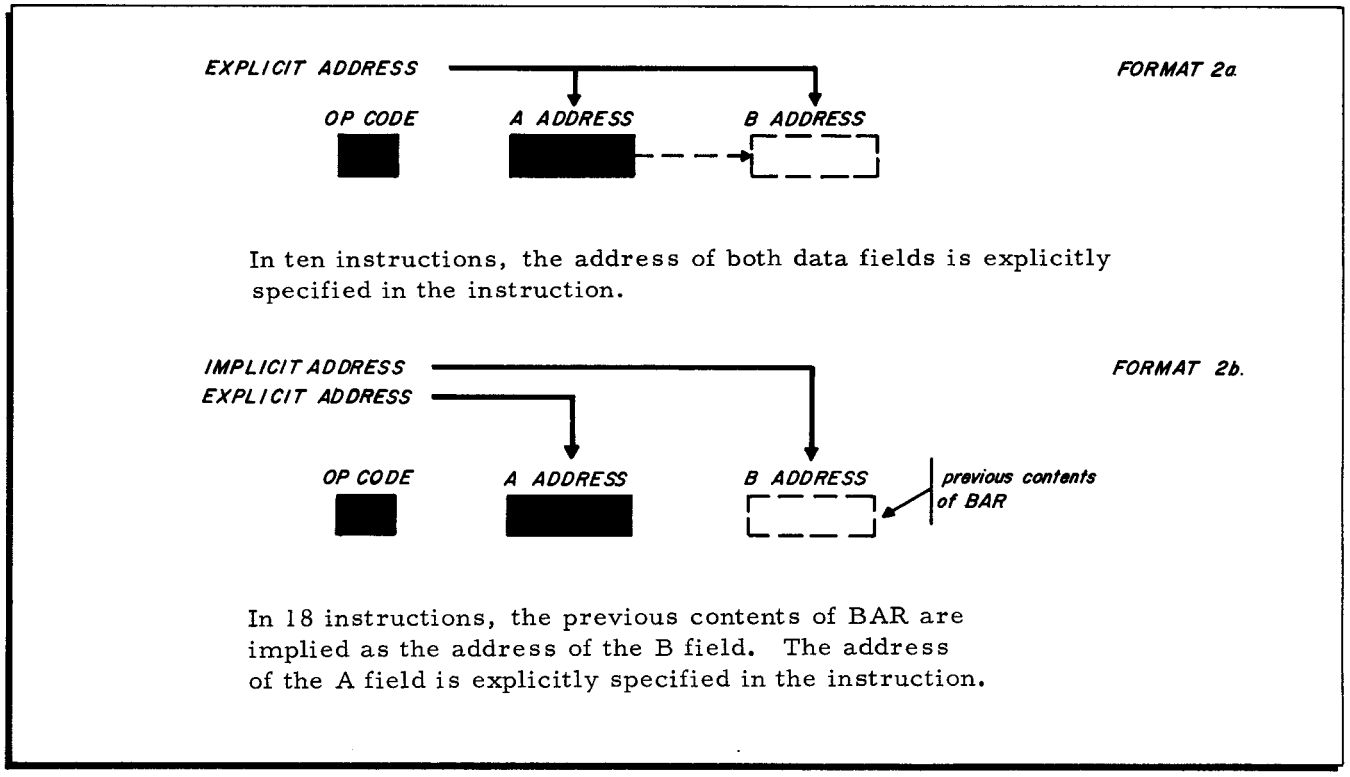


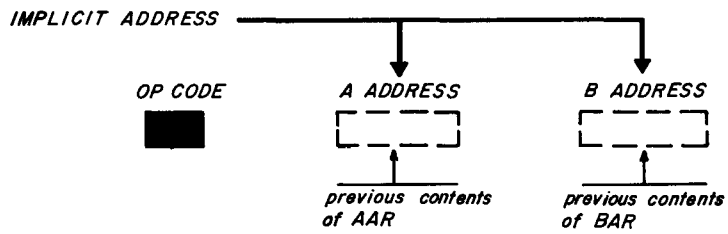
Figure 4-7. Series 200 Instruction Format 2

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 4-8).

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

[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.

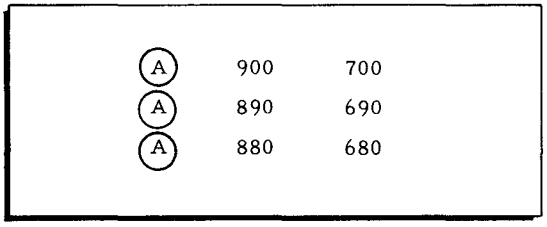
addressing reduces both the time required to perform the operations and the number of memory locations required to store the instructions.



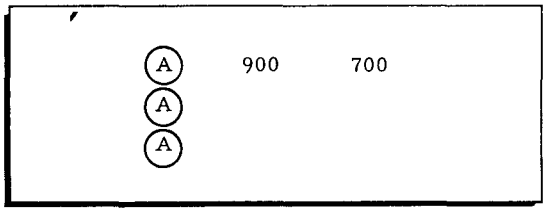
The addresses of both data fields are implied in the instruction.

Figure 4-8. 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.



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 Types 201-2, 1201, and 2201 processors, variant characters can be chained by an instruction coded in any format (i.e., format 1, 2, or 3 shown on page 4-14). The previous contents of the variant register are not disturbed by the processing of an instruction which does not contain a variant character.

In the Types 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 4-14).

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 8) 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

ALSO REFER TO ADDENDUM

INTRODUCTION

The preparation of Series 200 programs is greatly simplified by 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.

The Easycoder system consists of two basic elements: the Easycoder symbolic language and the Easycoder Assembly Program. The Easycoder language is used to write the symbolic program (the source program), while the Assembly Program translates the source program into the actual machine-language program (the object program).

To prepare a program in Easycoder language, the programmer uses an Easycoder Coding Form (see Figure 5-4) 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. 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 the Easycoder Assembly Program.

The Assembly Program 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. A secondary output of the Assembly Program is a complete printed summary of the symbolic source program and the corresponding machine-language entries.

If the programmer finds it necessary to alter the object program after it has been assembled, he can isolate the affected areas in the source program, enter the required changes, and then use the Assembly Program to reassemble a corrected version of the object program. Figure 5-1 illustrates the relationship of the source program, the Assembly Program, and the object program.

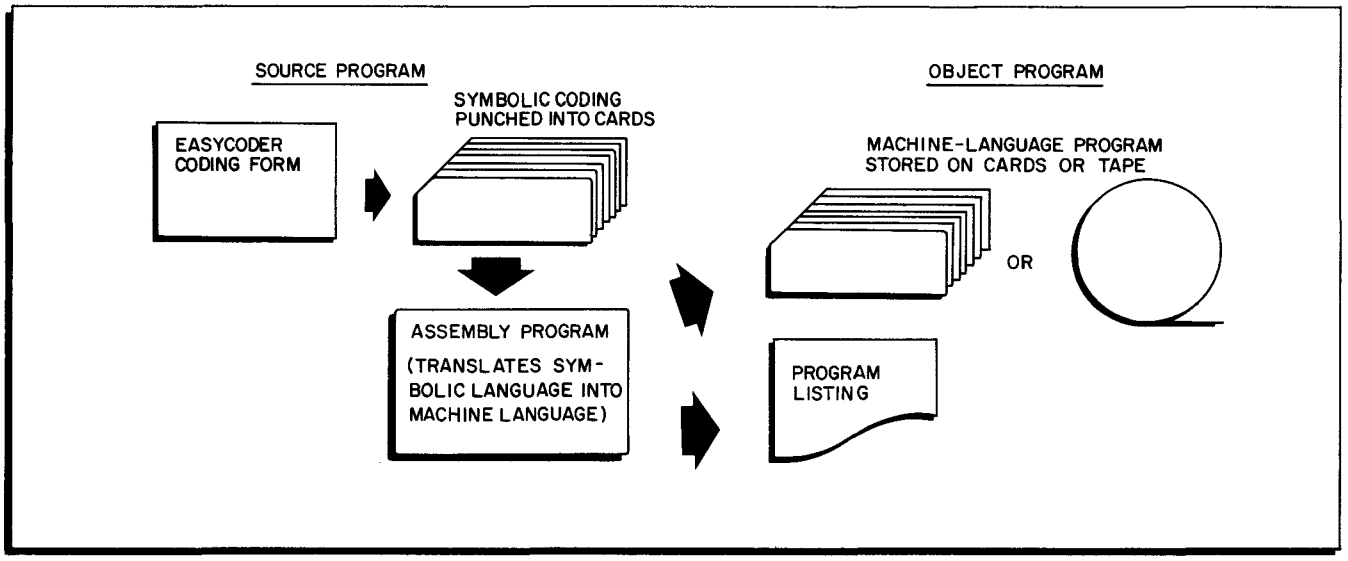


Figure 5-1. Relationship of Source, Assembly, and Object Programs

EASYCODER SYMBOLIC LANGUAGE

The 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 completely described in Section 6.
2. Assembly control statements are used by the programmer to control the assembly of his program. A complete description of assembly control statements can be found in Section 7.
3. Data processing statements are the actual machine instructions to be executed in the object program. Section 8 contains a description of the data processing statements employed by the Models 200, 1200, and 2200.

EASYCODER ASSEMBLY PROGRAM

The Easycoder Assembly Program 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 Assembly Program also produces a printed listing containing the symbolic source program and the corresponding object program entries. (See Figures 5-2 and 5-3.)

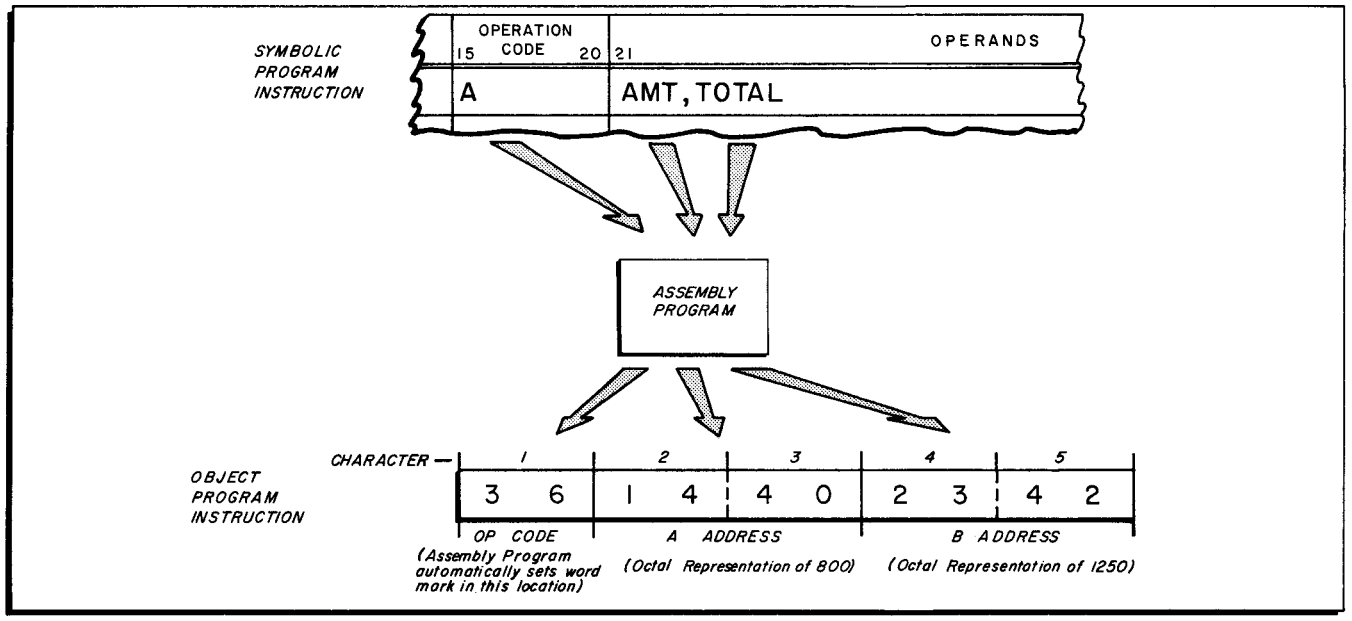


Figure 5-2. Two-Character Address Assembly

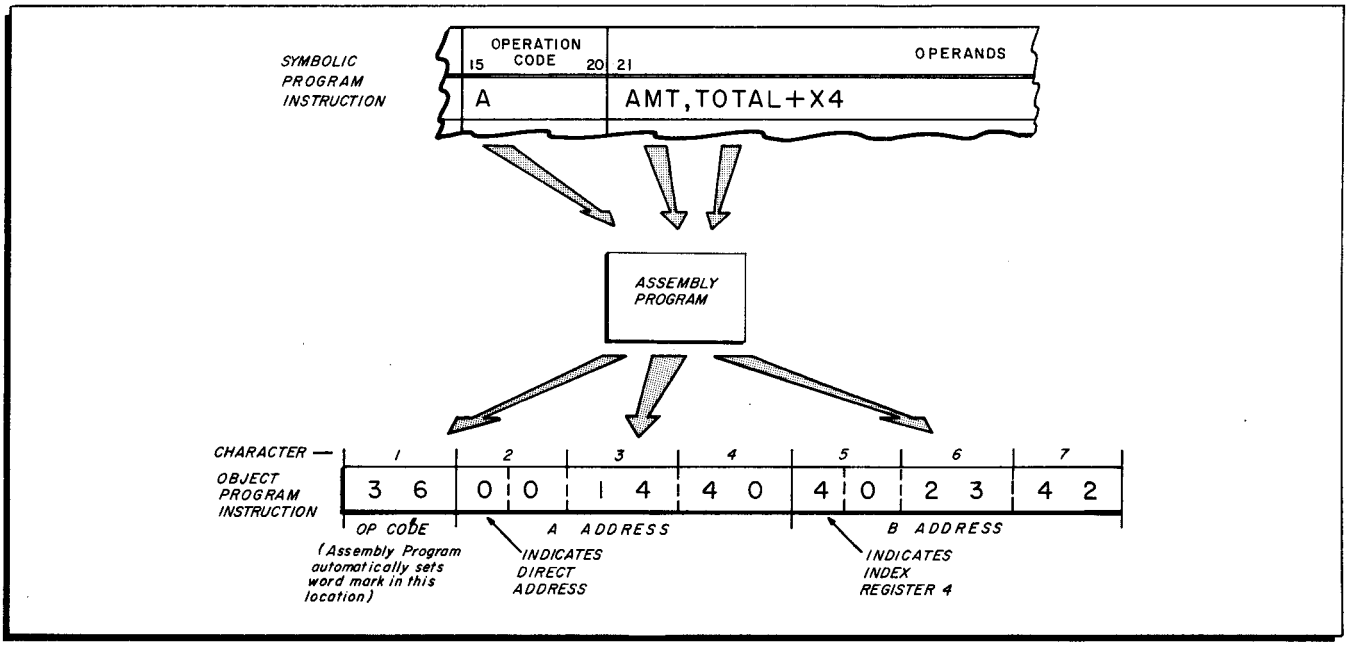


Figure 5-3. Three-Character Address Assembly

Figure 5-2 illustrates how the Assembly Program 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 Assembly Program assembles an object program instruction using three-character address assembly. Assume that the tags are assigned the same values as in Figure 5-2.

CODING FORM

Programs are written on the Easycoder Coding Form (Figure 5-4). 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 as to be punched.

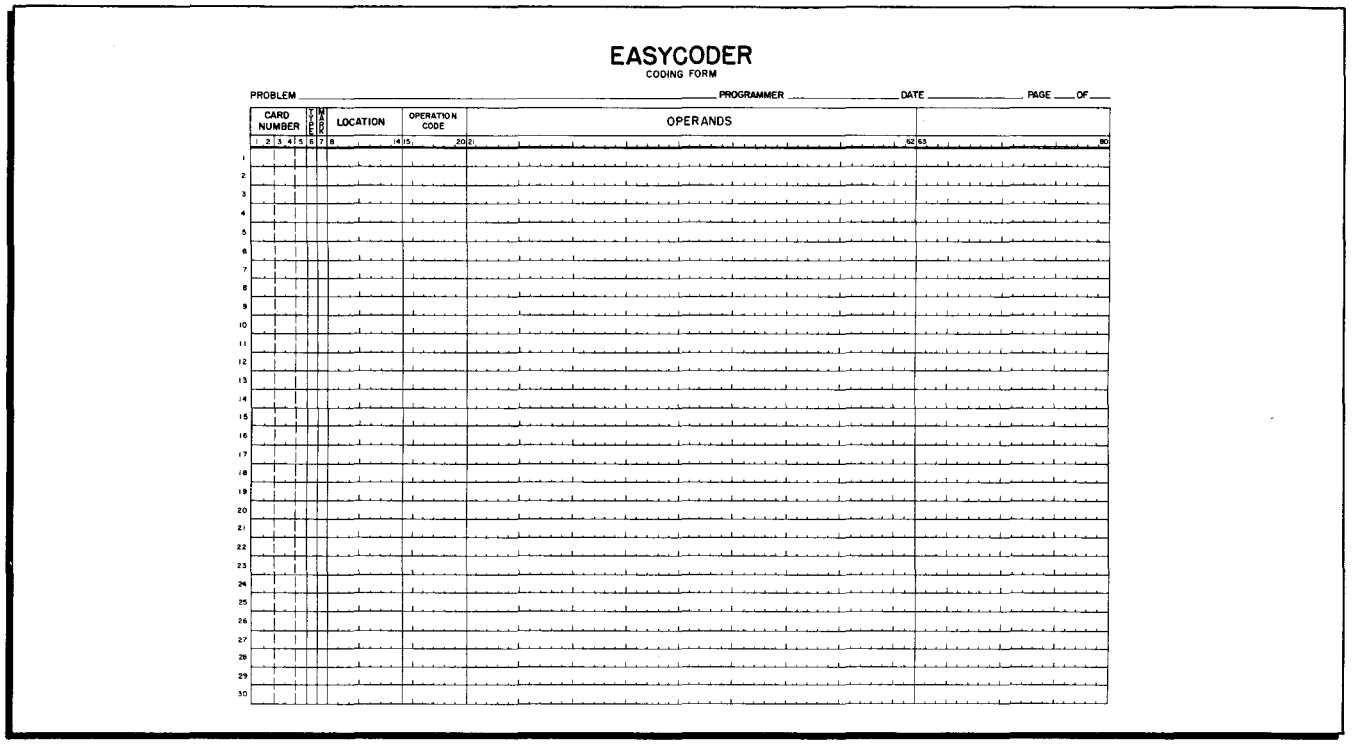
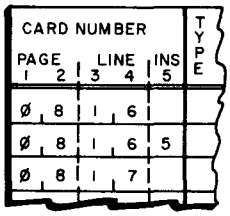


Figure 5-4. Easycoder Coding Form

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.



TYPE (Card Column 6)

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 appears in the program listing but does not appear as an entry in the object program.

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
	*		SPECIFY	CONTROL CONSTANTS		

MARK (Card Column 7)

This field, used in conjunction with data formatting operations (described in Section 6), serves to set up required punctuation. Two sets of punctuation indicators are available: set I may be employed with all Easycoder Assembly Systems; set II, however, may only be used with the Easycoder C Assembly System.[1] Both punctuation sets are described below.

Set I, consisting of a blank, 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 C Assembly System, 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.

[1] See Section 7 for definitions of the various Easycoder Assembly Systems.

Table 5-2. Set II Punctuation Indicators
(Easycoder C 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 a symbolic tag or an absolute memory address, or it can be left blank. Symbolic tags provide meaningful symbolic references for storage locations, constants, and instructions that are referenced 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.

A symbolic tag is composed of from one to six characters which can be alphabetic (A to Z) or numeric (0 to 9); the first character of the tag must be alphabetic. Location field entries are normally left-justified; that is, the first character is written in column 8. If a symbolic tag is assigned to an instruction, the address assigned to the tag by the Assembly Program will be the address of the operation code (the leftmost character in the instruction). If a tag is assigned to a constant or a reserved area, the address assigned to the tag will be that of the rightmost character in the field. (These address assignment conventions can be reversed by leaving column

8 blank and entering the first character of the tag in column 9. In this case, a tag assigned to an instruction will refer to the rightmost character of the instruction, and a tag assigned to a constant or reserved area will refer to the leftmost character in the field.)

The programmer may also write an absolute memory address (expressed as a decimal number) in the location field. This address will be assigned to the operation code of an instruction, or to the rightmost character of a constant or a reserved area. (If the address is written starting in column 9, the address assignment conventions are reversed as described above.)

EASYCODER
CODING FORM

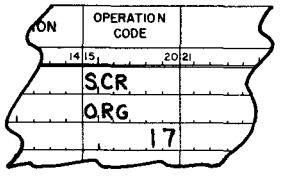
PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
	1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62 63 ... 80
1				BEGIN	MCW	FICA,TAX
2					B	BEGIN
3				DATE	DCW	@06/05/65@

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.

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.



OPERANDS (Card Columns 21-62)

The operands field is a variable-format field which can contain a series of entries separated by commas and terminated by the first blank following any character other than a comma or a blank. Column 62 also terminates the operands field. Any punches appearing in columns 63-80 of any line other than a remarks line (see page 5-5) are ignored and do not even appear in the object program listing. [1]

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. As explained in the following paragraphs, relative, indexed, and indirect addressing can be used in conjunction with absolute or symbolic addresses.

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	1				A	50,TOTAL	
2							
3	2				BCT	EQUAL,42	
4							
5	3				ZA	TOTAL,TMP+X3	
6							
7	4				MCW	TOTAL-7+X6,GROSS	
8							
9	5				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 3.

[1] The Easycoder C Assembly Program (see Section 7) interprets punches in columns 63-80 as comments and prints these comments in the program listing.

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 6. 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 6 contains a value of 80, the resultant A 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 (*) 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
				MCW	32,4000

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	

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 Assembly Program 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	
4						
5						

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+100,AMTPD+ERROR-10	

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, INST+3, or INST+4, depending on whether two-, three-, or four-character address assembly were specified.

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
			INST	A	SUBT,TOTAL	

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 the data) to be operated on by an instruction. All literals, except binary literals, can be coded with a maximum length of 40 characters. A binary literal can be coded with a maximum length of six characters.

The Assembly Program 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 Assembly Program generates a constant containing the value of the literal, with a word mark in the left-most 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. A constant that exceeds five characters 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 Assembly Program 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
				S	+24, ACCUM	

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.

[1]Available only with Easycoder B and C.

Binary Literals

A binary literal is represented as a decimal entry in the operands field of a symbolic instruction. The Assembly Program 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 (not to exceed six) used to store this value.

A binary literal is coded by writing a # sign, followed by a number from 1 to 6 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 Assembly Program will store the binary literal in twos-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 2688 (expressed as an 18-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	#2B50,BEGIN+2	
2			BEGIN	MCW	ITEMA,TOTAL	
3						
4				MCW	#3B2688,IND7	
5						

Octal Literals

Octal literals are coded in octal notation (see Appendix A). The programmer must specify the number of six-bit characters (not to exceed 20) 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 Assembly Program 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
1				HA	#3C7777,MASK

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

Alphanumeric Literals

An alphanumeric literal can contain blanks, decimal, alphabetic, and special characters (excluding the @ symbol). It is specified by writing the @ symbol before and after the value of the literal. If the @ symbol is required within a literal, a DCW statement (see page 6-2) should be used.

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
1				MCW	@ACCOUNTS PAYABLE,8/10/65@,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.

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
1				MCW	WAGE,TEMP#5
2					

In the instruction above, the entry TEMP#5 causes the Assembly Program 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 Assembly Program 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 or a minus sign.

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 two 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	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1				MCW	+AMT,MODIF+2
2			MODIF	A	Ø,TOTAL

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 variant characters are listed in Appendix B.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1				BCT	OFLOW,SØ
2				BCC	NEG,SUM,Ø6

[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 the Easycoder A or B Assembly System) or from one to four (using the Easycoder C Assembly System). 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 7-11).

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 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 8). 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 aplhanumeric 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), 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 and the decimal number of the desired index register (see Tables 4-2 and 4-3, pages 4-11 and 4-13, respectively).

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.

[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 the Easycoder A or B Assembly System) or from one to four (using the Easycoder C Assembly System). 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 7-11).

However, the programmer may use the symbolic address if he equates it to the absolute address using an EQU statement (see page 7-16).

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	DATA+X6,POS
2						
3					BA	STORE,12
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 6 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 12. The use of the symbolic designation X12 implies that an EQU statement was used to equate it to the absolute address of index register 12. 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 one (i.e., if index register one contains 126, the effective A address is 120).

Three- or four-character address assembly must be specified (see ADMODE, page 7-9) whenever indexed addressing is to be performed. When the Assembly Program translates an indexed address into a machine-language entry (see Figures 5-5 and 5-6), the translated index register designator is automatically inserted into the address modifier bits of the assembled address.

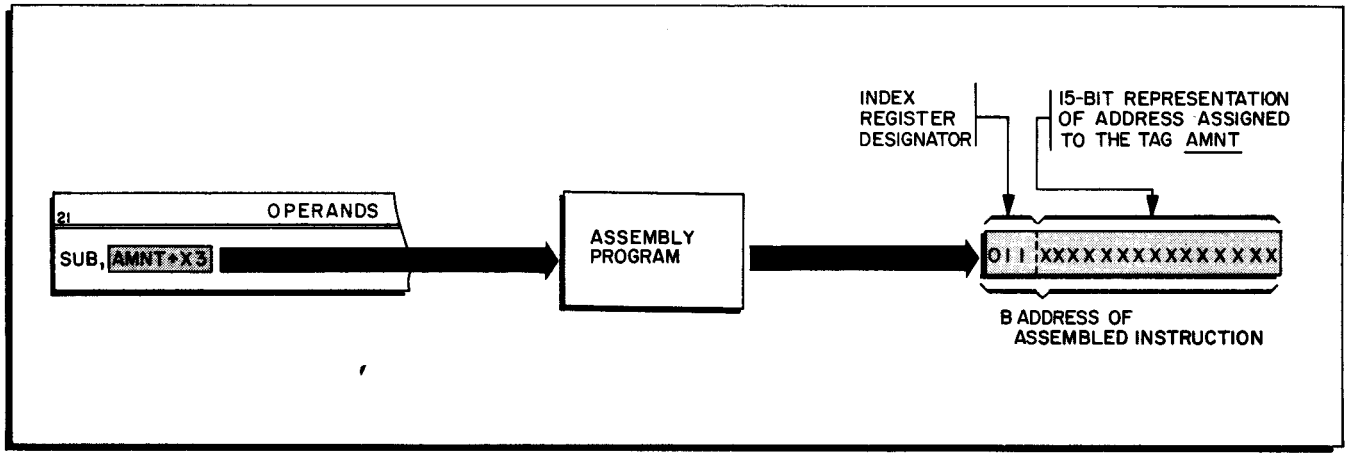


Figure 5-5. Assembly of Indexed Address in Three-Character Addressing Mode

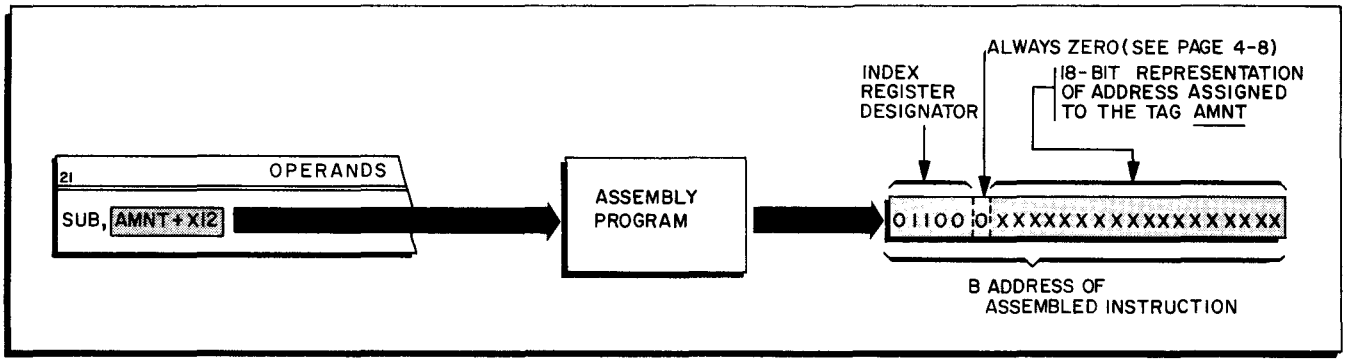


Figure 5-6. Assembly of Indexed Address in Four-Character Addressing Mode

INDIRECT

An indirect address is specified by enclosing the address (either symbolic or absolute) in parentheses. For example, in the sample instruction below, the parentheses around the tag DATA indicate to the Assembly Program 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.

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
				MCW	(DATA),WORK

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.

Three- or four-character address assembly must be specified whenever indirect addressing is to be used. When the Assembly Program translates an indirect address into a machine-language entry (see Figures 5-7 and 5-8), an octal value of 7 (three-character mode) or 20 (four-character mode) is automatically inserted into the address modifier bits of the assembled address.

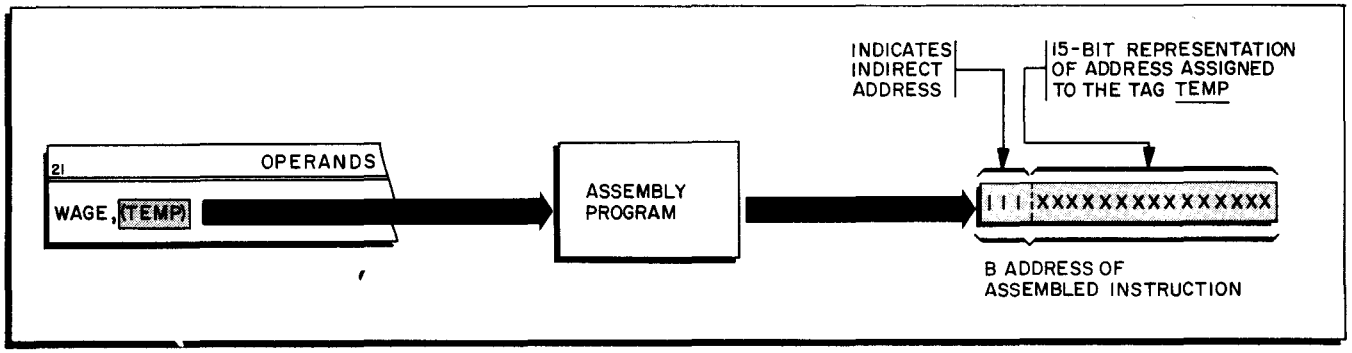


Figure 5-7. Assembly of Indirect Address in Three-Character Addressing Mode

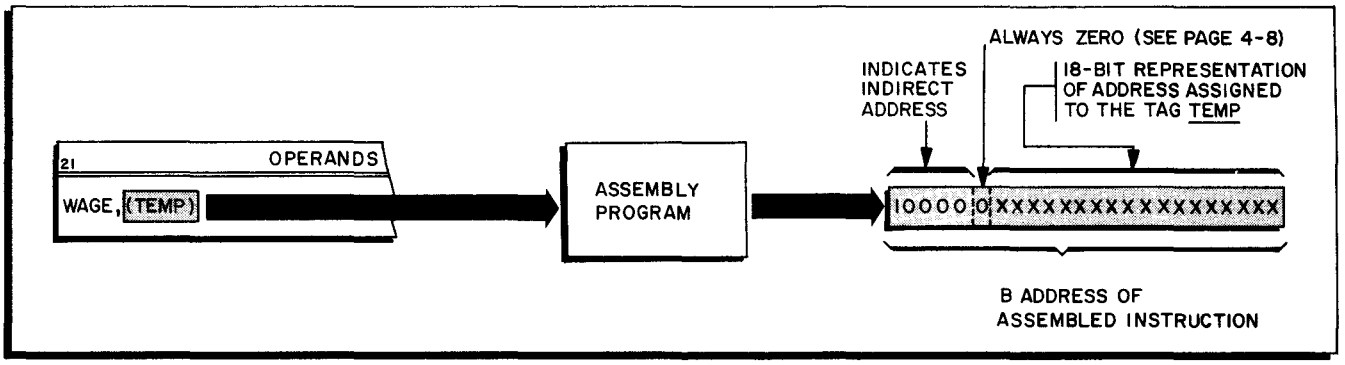


Figure 5-8. Assembly of Indirect Address in Four-Character Addressing Mode

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 may be employed only with the Easycoder B and C Assembly Systems (see page 7-1).

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 the Assembly Program. Instead they are treated as definitions which cause the Assembly Program 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 Assembly Program. In storing the constant, the Assembly Program automatically sets a word mark in the leftmost character position of the storage field. Item marking may be specified as in Table 5-1 (page 5-5). An L in column 7 thus results in a record mark with a DCW statement.

NOTE: If the Easycoder C Assembly System 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 5-6).

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. 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.

Decimal Constants

Signed decimal constants are specified by writing a plus or a minus sign in column 21 followed by the value of the constant. When the constant is assigned to a storage field, the Assembly Program places the sign in the zone bits of the rightmost character of the constant.[1] Unsigned decimal constants are written in the operands field beginning at column 21.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

CARD NUMBER	TYPER	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5 6 7			8 ... 14	15 ... 20	21 ... 62	63 ... 80
1			DEC	DCW	+22	
2			HAL	DCW	45	

The first statement above shows the decimal value of +22 defined as a decimal constant. The second statement defines the unsigned decimal value 45 as a constant which can be referred to via the tag HAL.

[1] See the description of sign codes beginning on page 8-9.

Binary Constants

A binary constant is actually written as a decimal entry which is then automatically converted to a binary value by the Assembly Program. 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 column 21); (2) a number from 1 to 6 which designates the number of six-bit characters needed to store the resulting binary value; (3) the letter B; (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 Assembly Program stores the binary constant 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			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 column 21); (2) a number (not to exceed 20) 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 Assembly Program 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			OCT7	DCW	#2C7777

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, +, -, #, or the digits 0 through 9 (see the second entry below).
3. A number sign (#) is followed by a number from 1 through 40 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	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62	63 ... 80
1			COST	DCW	@$2,128.60@	
2						
3			RATE	DCW	%@SIX DOLLARS/HR%	
4						
5			DATE	DCW	#4A1965	

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 column 21) followed by a decimal value (from 1 to 40) which indicates the number of blank storage positions desired.

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			BLANK	DCW	#21	

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

[1] This third method of coding alphanumeric constants is applicable only when using the Easycoder C Assembly System (see page 7-1).

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 5-5).

NOTE: If the Easycoder C Assembly System 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 5-6).

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 Assembly Program), the RESV statement does not alter the contents of the area defined when used with the Easycoder A or B Assembly System. Rather, it simply sets aside a storage area to which the programmer can refer by a symbolic tag. If it is desired to clear the reserved area to zeros in either of the above systems, the CLEAR statement must be employed (see page 7-15). The number of characters in the reserved area must be specified in the operands field of the RESV statement. A previously defined tag may be written in the location field.

When used with the Easycoder C Assembly System, 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 column 20 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 area. However, a punctuation indicator from set I may be placed in column 7 (see page 5-5). In addition, if the Easycoder C Assembly System 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 5-6).

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1			STORE	RESV	3Ø
2			CARD	RESV,Ø	8Ø
3					

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, 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 5-5. 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 the Easycoder C Assembly System 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 5-6).

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	
4							

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

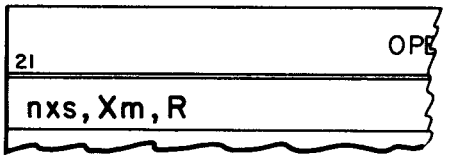
[1] The Define Area statement may be employed only with the Easycoder B or C Assembly System.

areas are identical in format. In other words, the programmer uses a DA statement to provide the Assembly Program with the following basic information:

1. The number (N) and size (S) of the reserved area(s).
2. The index register (Xm) to be associated with each reference to a field or subfield within the reserved area(s) (optional).
3. The character R which will place a record mark one position to the right of the rightmost reserved area (optional).

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.

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



If a single 80-character area is to be defined, the value of the 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)[1] following the area definition. All references to the fields 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.

[1] Index registers 1 through 6 are used with Easycoder B, while index registers 1 through 15 can be used with Easycoder C.

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 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 5-5).

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 5-5).

NOTE: The list of punctuation indicators specified in set II (page 5-6) may not be used with DA statements.

The Assembly Program 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	
7						
8						

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 two.
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 two.)

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.

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 number of assembly systems are available to the Series 200 user. These systems include:

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.)

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.

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.)

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.)

A summary of the assembly control statements available with the Easycoder A, B, and C Assembly Systems, 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 (by shading) the assembly systems that may use that particular statement.

Table 7-1. Assembly Control Statements

Easycoder A		Easycoder B		Easycoder C	
Assembly Control Statements	Page Ref.	Assembly Control Statements	Page Ref.	Assembly Control Statements	Page Ref.
Program Header	7-3	Program Header	7-3	Program Header	7-3
				Segment Header	7-3
Execute	7-4	Execute	7-5	Execute	7-5
Origin	7-6	Origin	7-7	Origin	7-7
Modular Origin	7-7	Modular Origin	7-7	Modular Origin	7-7
		Literal Origin	7-8	Literal Origin	7-9
Admode	7-9	Admode	7-9	Admode	7-10
Equals	7-10	Equals	7-10	Equals	7-11
Control Equals	7-11	Control Equals	7-11	Control Equals	7-12
Memory Dump	7-12				
				Skip	7-13
				Suffix	7-13
				Repeat	7-14
				Generate	7-14
Clear	7-15	Clear	7-16	Clear	7-17
End	7-17	End	7-18	End	7-19

Program Header
PROG

A	B	C

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 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 8K Operating Procedures, DSI-406.)

NOTE: When BRT format is specified, a segment number of 01 is generated by the Assembly Program for the first segment (memory load) following the program header. If Execute statements (see page 7-4) 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 the Easycoder C Assembly System, 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 Assembly C Operating Procedures (DSI-315A) for a detailed description.

Segment Header
SEG

,

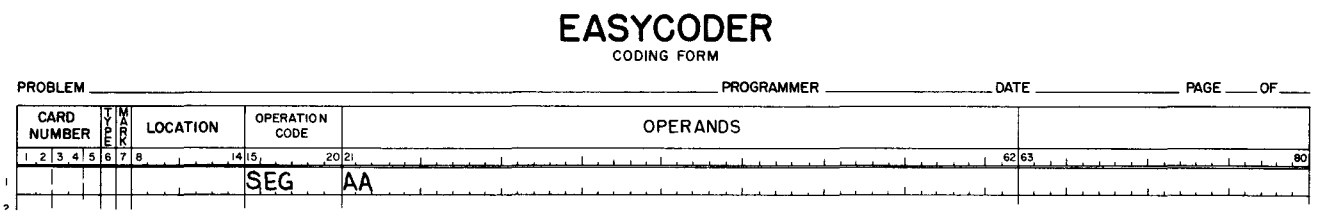
A	B	C

Programs written for Easycoder C assembly 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 and each Execute statement. If it is desired to omit this statement, it must be omitted from the entire program; in this case the assembly program generates segment identifications (starting with 01).

EASYCODER C

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.



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

The end of a memory load is indicated by an EX statement. When the coding inserted by the assembly program 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	SEG3	
2						

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 SEG3 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.

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

See the sample statement given above for Easycoder A.

EASYCODER C

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 this assembly system, the EX statement enables a program to be loaded and executed one segment at a time. Each segment except the last must end with this statement.

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., the PLUS – Card Loader-Monitor bulletin, DSI-349, or the PLUS – Tape Loader-Monitor bulletin, DSI-327).

See the sample statement given above for Easycoder A.

Origin
ORG

A	B	C

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 Assembly Program 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 location 0, an ORG statement must be inserted into 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	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
	1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62
1					ORG	750
2						
3					ORG	ORTAG
4						

The first statement above indicates to the Assembly Program that all subsequent entries should be assigned sequential addresses beginning with location 750. The second statement directs the Assembly Program 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

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. — 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: 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.

See the sample statements given above for Easycoder A.

Modular Origin MORG

A	B	C

The modular origin statement is similar to the ORG statement described above. The MORG statement indicates to the Assembly Program 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, B, and C

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
1				MORG	32
2					
3					

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

Literal Origin
LITORG

A	B	C

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

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); this address is assigned the tag, if any, in the location field. 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.

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
1			LIT	LITORG	1550
2					
3					

In the LITORG statement above, the Assembly Program is directed to assign sequential

addresses — starting with location 1550 — to all previously encountered literals. This instruction is also tagged LIT. (Note that the tag begins in column 8; it must not be indented.

EASYCODER C

The op code field must contain the letters LITORG, while the operands field contains an address (either absolute or symbolic); this address is assigned the tag, if any, in the location field. 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 1,340 (decimal) must not be used.

See the sample statement given above for Easycoder B.

Set Address Mode
ADMODE

A	B	C

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 8-69). 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	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					ADMODE	2	
2							
3					ADMODE	3	
4							

The Assembly Program, 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 Assembly Program to change to three-character address assembly.

EASYCODER C

The letters ADMODE are placed in the op code field. The operands field contains a 2, 3, or 4 to denote whether all subsequent instructions are to be assembled in the two-, three-, or four-character addressing mode. If an ADMODE statement is not included at the beginning of the source program, three-character addressing is assumed by assembly. (It should be a general rule, however, to include an ADMODE statement at the outset of every program.)

See the sample statements given for Easycoder A and B.

Equals
EQU

A	B	C

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-20	
4							

The first EQU statement above causes the Assembly Program 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 Assembly Program will assign the tag QUAN to the location specified by address arithmetic as AMT-20.

EASYCODER C

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

The CEQU statement assigns the symbolic tag written in the location field to the octal 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.

The octal value written in the operands field (although coded as an octal constant) is still treated as an assembly definition. Consequently, it does not appear as an object program entry.

EASYCODER A and B

The location field contains a symbolic tag, while the op code field contains the letters CEQU. The operands field contains the 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
3						

The sample coding above illustrates how a symbolic tag can be used in place of a variant character. The CEQU statement directs the Assembly Program 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

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

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

Memory Dump
HSM

A	B	C

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

The Assembly Program normally single-spaces an assembly listing and skips 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

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 Assembly Program 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
				SKIP	9

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

Suffix
SFX

A	B	C

The SFX statement directs the Assembly Program 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

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					SFX	E
2				TOTAL	A	FICA+TOTAX-2ϕ
3						
4						

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

Repeat
REP

A	B	C

This statement, used with the constant-defining statements DC and DCW, directs the Assembly Program to repeat the following 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 Assembly Program repeats the statement without variation.

EASYCODER C

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					REP	6
2				OCTS6	DCW	#2C6
3						
4						

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

Generate
GEN

A	B	C

This statement directs the Assembly Program 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

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
5					

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.

Clear
CLEAR

A	B	C

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

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

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

The last source program instruction must be the END statement, which indicates to the Assembly Program 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 Assembly Program 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. If the operands field is blank, the machine halts after the load operation has been completed.

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
				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 Assembly Program 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 assembly program 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. If the operands field is blank, the machine halts after the load operation has been completed.

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

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

   c. 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. If the operands field is blank, the machine halts after the load operation has been completed. When BRT format is specified, all other fields of the END instruction are ignored by the Assembly 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				MAL	END	OBJECT	
2							
3					END	OBJECT	
4							

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

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
				END	STARTL

The sample END statement above indicates to the Assembly Program 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 lefthand 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 the formats which may be used when coding the instruction.

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

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

Timing: The formulas to be used in calculating the timing of the instruction (in memory cycles) are presented. These formulas are for instructions using direct addressing. If address modification is to be used, the formulas 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.

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.

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)
	Multiply
Z_{ta}	Number of trailing zeros (i.e., consecutive low-order zeros) in the A field
N_{mr}	Number of digits in the multiplier
Z_{mr}	Number of zeros in the multiplier
s	Sum of all multiplier digits
SUM	The sum of the upwards-rounded values of all multiplier digits divided by 2 (see note)
	Divide
Z_{la}	Number of leading zeros in the A field
Z	0 if $Z_{la}=0$; 1 if $Z_{la}\neq 0$
Z_{ld}	Number of leading zeros in the dividend
N_{dd}	Number of digits in the dividend
N_q	Number of digits in the quotient ($=N_{dd}-Z_{la}-N_a+Z_{la}+1$)
	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)

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

SYMBOL	MEANING
	Move Characters and Edit
Z	Number of characters scanned during zero suppression
$	Number of characters scanned during dollar sign insertion
	Store Variant and Indicators
N_s	Number of characters stored
N_j	Number of character locations bypassed to reach the next sequential op code
	Restore Variant and Indicators
N_r	Number of characters referenced
	Input/Output Instructions
N_c	Number of control characters in the instruction
N_{cn}	Number of control characters following control character 3 (C3)

NOTE: The value of SUM is derived in the following manner:

1. Divide each multiplier digit by 2.
2. Round off each result to the nearest (upwards) whole digit.
3. Add together the results arrived at in 2. for each multiplier digit.
4. The resultant sum = SUM.

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
(B) ADDEND + (A) AUGEND (B)	(B) MINUEND - (A) SUBTRAHEND (B)

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

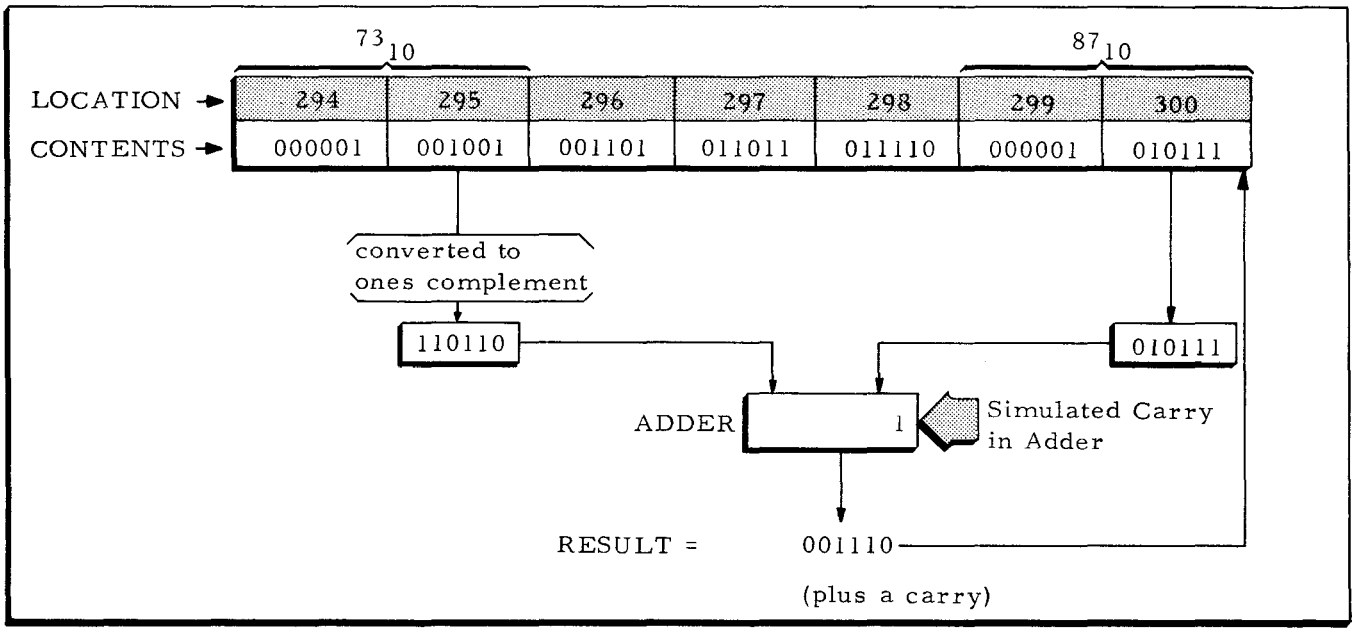
PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5 6	7	8	14 15	20 21	62 63 80
				BS	295,300

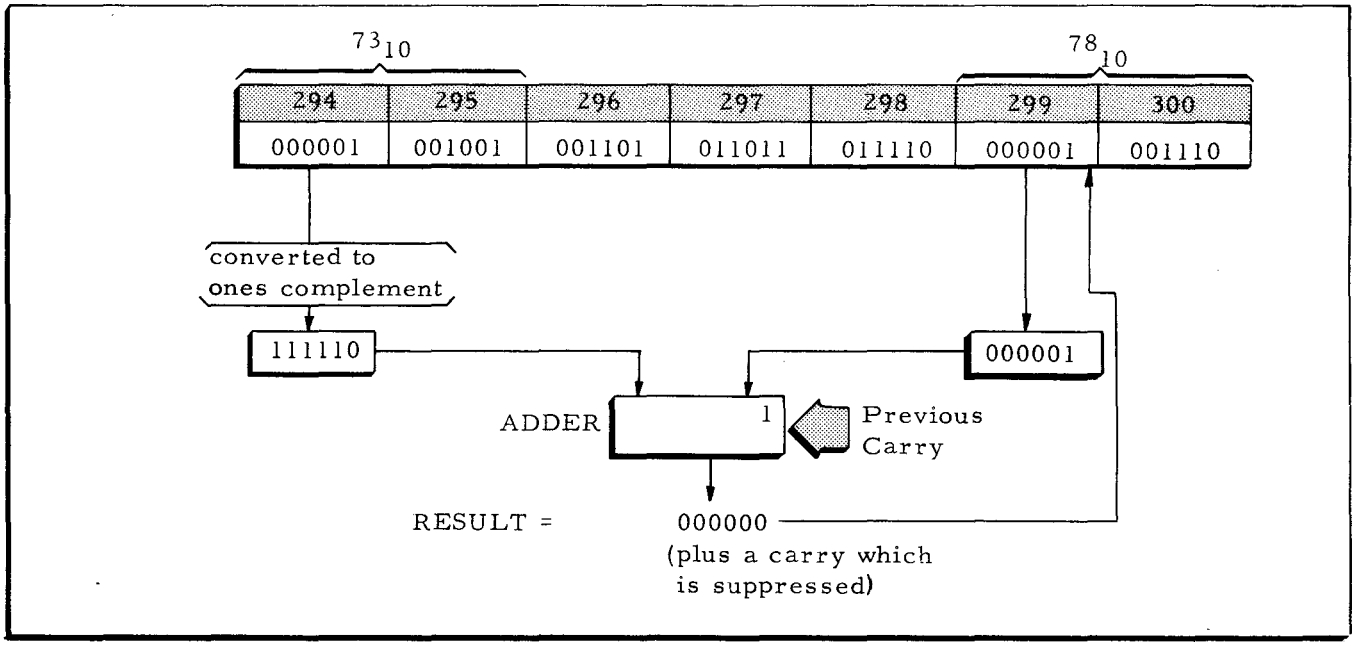
LOCATION →	294	295	296	297	298	299	300
CONTENTS → (binary)	000001	001001	001101	011011	011110	000001	010111

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 2-7). 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.

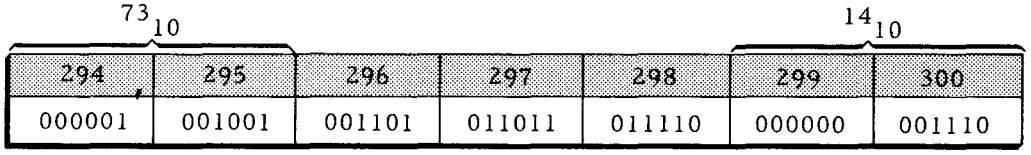


First Addition



Second Addition

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



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.

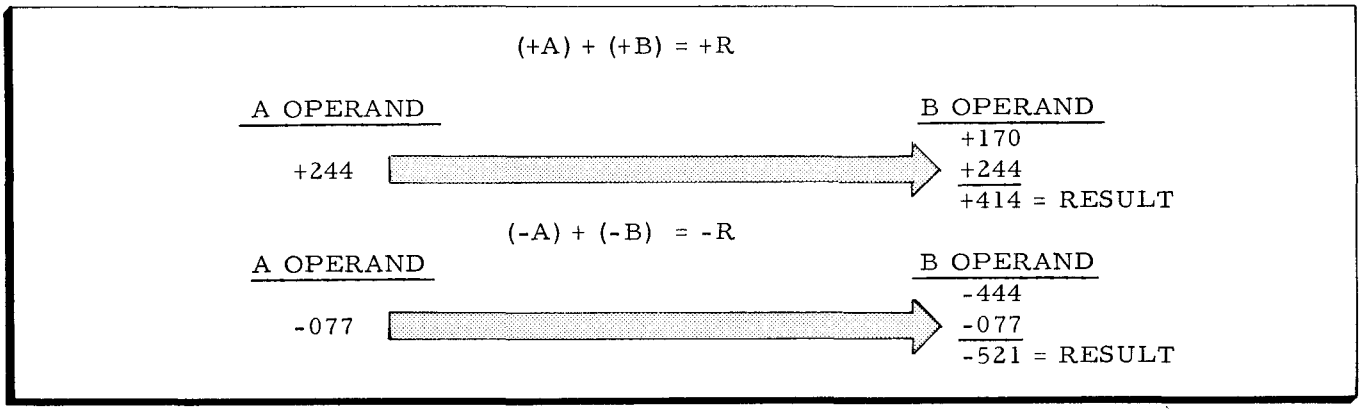


Figure 8-1. True Add Examples

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 without recomplementation.

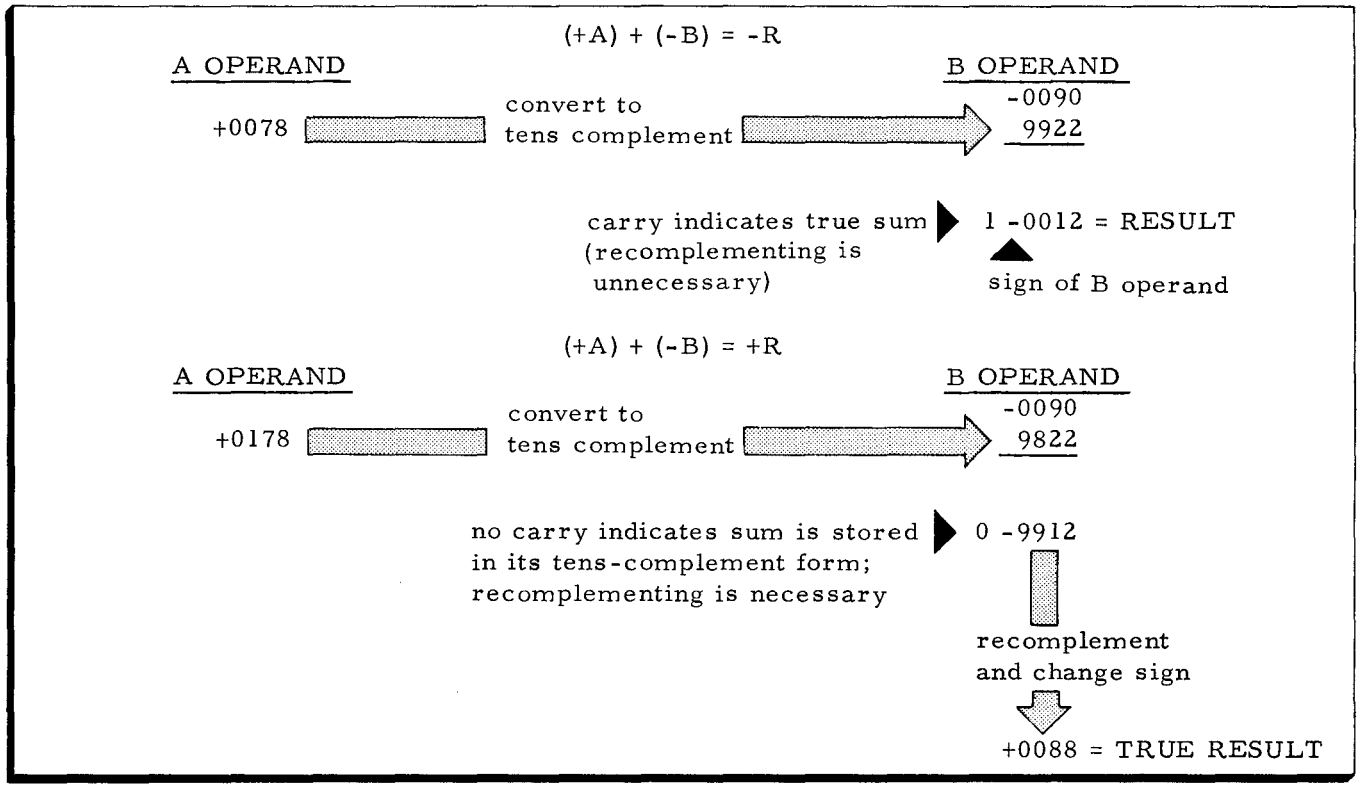


Figure 8-2. Complement Add Examples

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 greater value (- = 10, + = 01)
	-	+	Complement	
		-	True	-
SUBTRACT	+	-	True	-
		+	Complement	Normalized sign of the 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 8-41). 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).

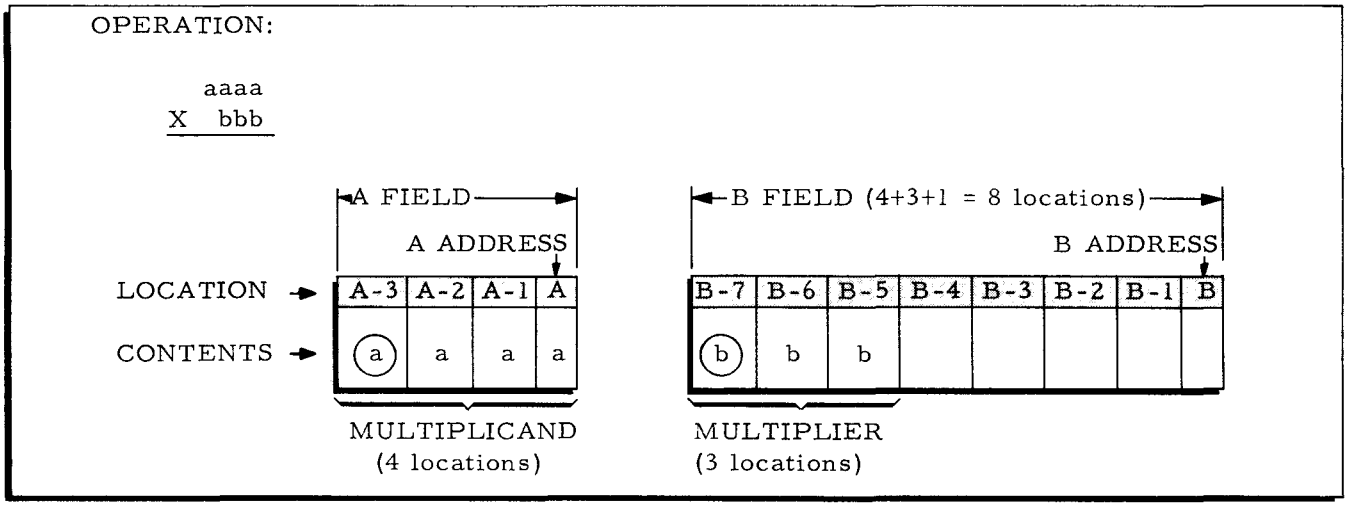


Figure 8-3. A and B Fields in Multiply Operation

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	+	+	-	-

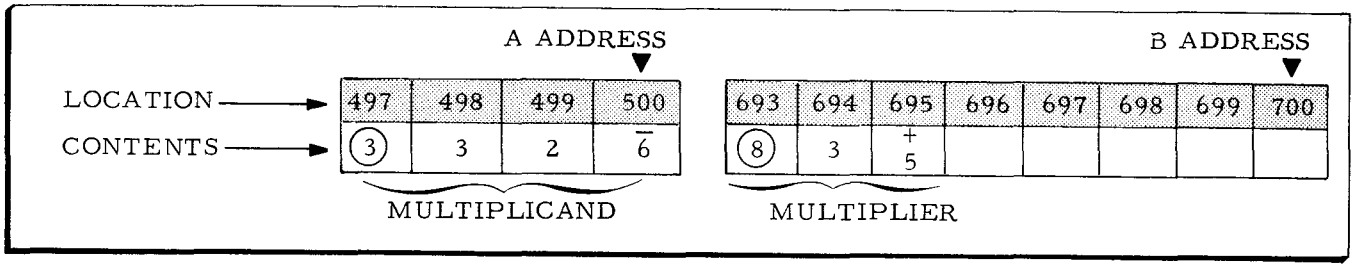
Consider the following Decimal Multiply instruction.

EASYCODER
CODING FORM

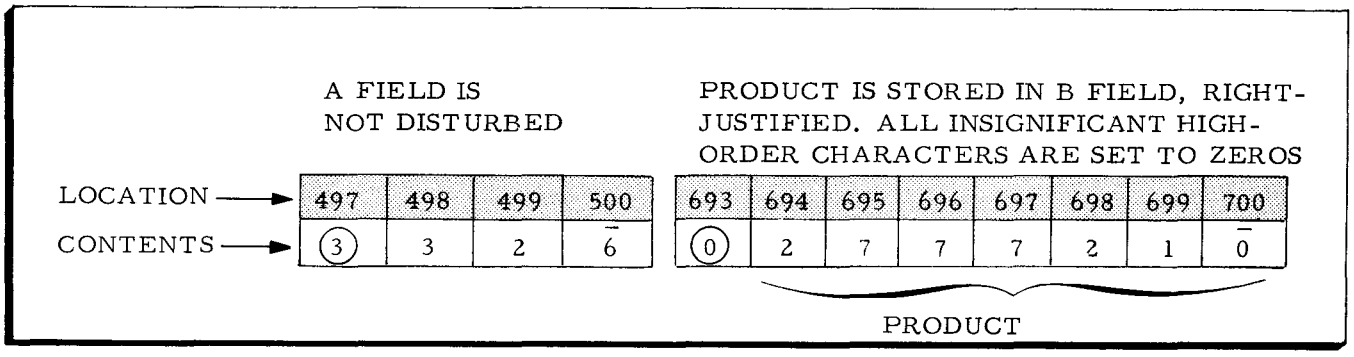
PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

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.



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.



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).

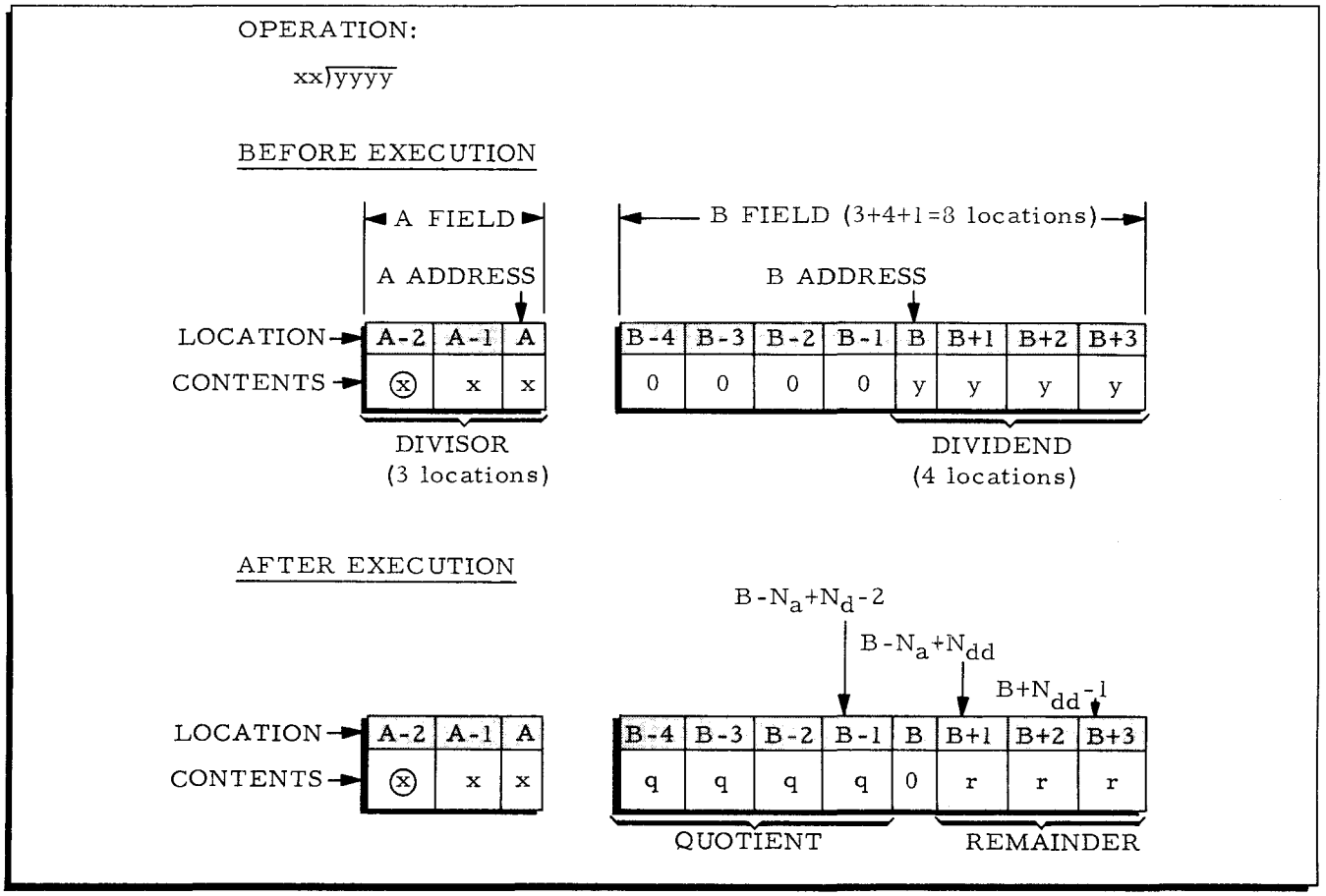


Figure 8-4. Factor Locations in Divide Operation

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, 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_d 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).

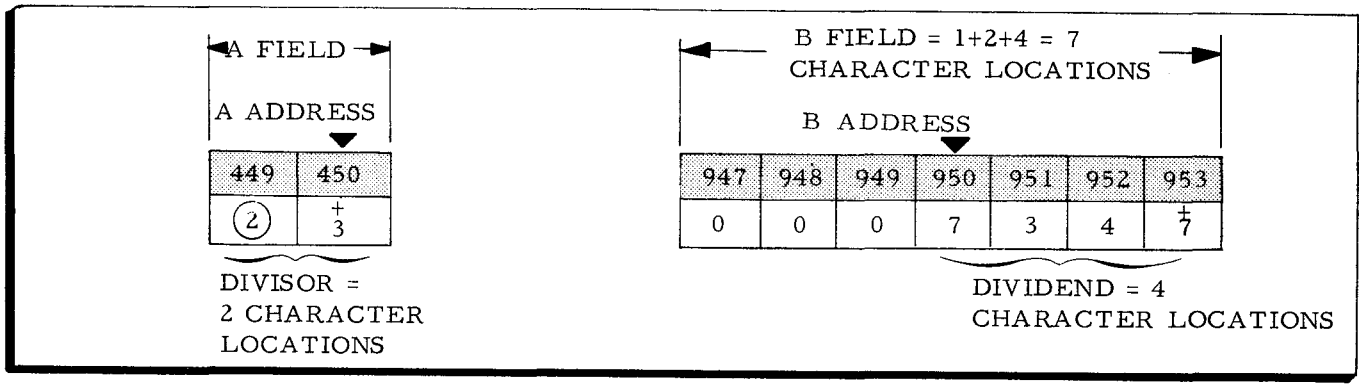
In the following example, the divisor is a two-character field whose rightmost location is location 450 and the dividend is a four-character integer whose leftmost location is location 950.

EASYCODER
CODING FORM

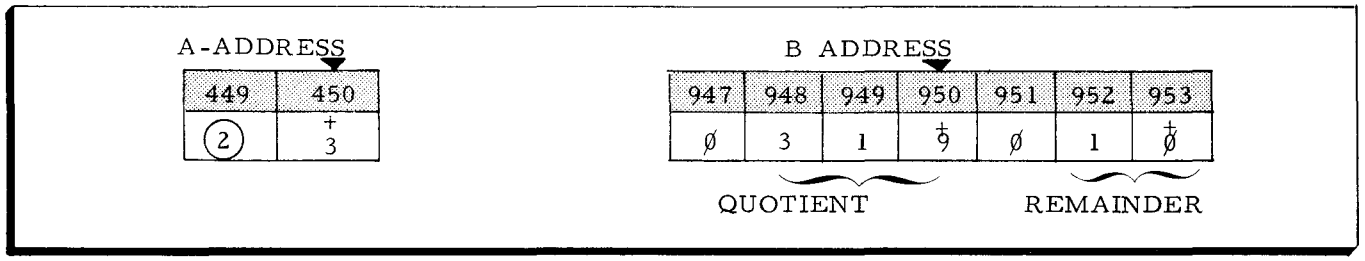
PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

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	450,950
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.

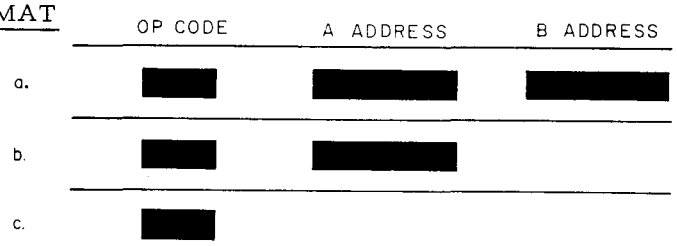


The quotient (+319) is stored in the leftmost character locations of the B field. The units position of the quotient (location 950) is equal to $B-N_a+N_{dd}-2$, or 950-2+4-2. The remainder is stored in the rightmost locations of the B field; its leftmost location (location 952) is equal to $B-N_a+N$,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



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.

TIMING

Format a: $T = N_i+2+N_w+2N_b$ memory cycles if no recomplement cycle is required.

$T = N_i+2+N_w+4N_b$ memory cycles if a recomplement cycle is required.

Format b: $T = N_i+2+3N_a$ memory cycles.

Format c: $T = 3+N_w+2N_b$ memory cycles if no recomplement cycle is required.

$T = 3+N_w+4N_b$ memory cycles if a recomplement cycle is required.

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_b-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. The two remaining cases (octal 12 and 13) are unspecified.

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

EXAMPLE

Add Bond Deductions to Total Deductions.

Description	Tag
Bond Deductions	BDED
Total Deductions	TDED

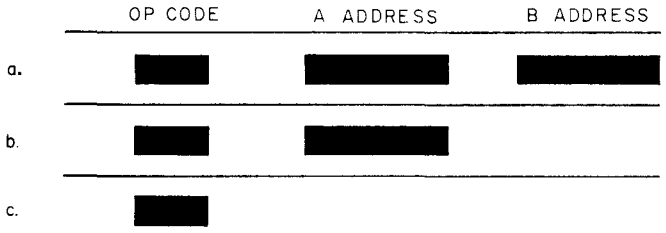
EASYCODER
CODING FORM

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
				A	BDED,TDED

S	SUBTRACT

FORMAT



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.

TIMING

Format a: $T = N_i+2+N_w+2N_b$ memory cycles if no recomplement cycle is required.

$T = N_i+2+N_w+4N_b$ memory cycles if a recomplement cycle is required.

Format b: $T = N_i+2+3N_a$ memory cycles.

Format c: $T = 3+N_w+2N_b$ memory cycles if no recomplement cycle is required.

$T = 3+N_w+4N_b$ memory cycles if a recomplement cycle is required.

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. The two remaining cases (octal 12 and 13) are unspecified.

4. The overflow and zero balance indicators are set by a subtract operation.

EXAMPLE

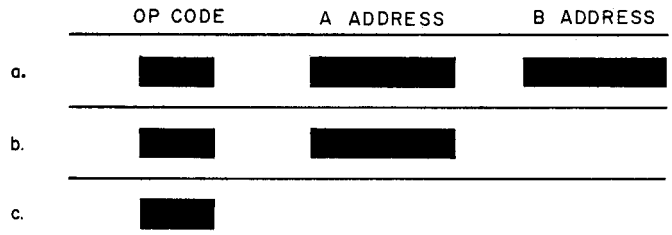
Subtract the contents of the five-character fields starting at location 940, 945, 950, and 955 from the contents of the eight-character fields starting at locations 648, 656, 664, and 672.

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
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.

TIMING

Format a: $T = N_i+1+N_w+2N_b$ memory cycles.[1]

Format b: $T = N_i+1+3N_a$ memory cycles.[1]

Format c: $T = 2+N_w+2N_b$ memory cycles.[1]

[1] Add one memory cycle to each of these times if the instruction is being executed in a Type 2201 processor.

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 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).

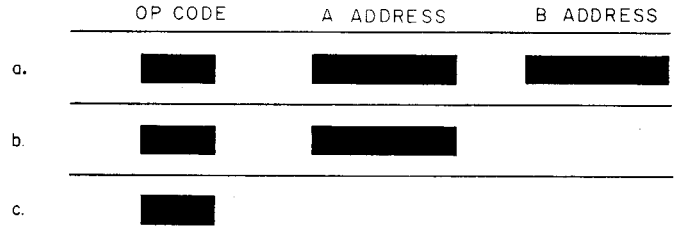
EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62 63 ... 80
				BA	B7+4, TEN

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 8-6). 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.

TIMING

Format a: $T = N_i+1+N_w+2N_b$ memory cycles.[1]

Format b: $T = N_i+1+3N_a$ memory cycles.[1]

Format c: $T = 2+N_w+2N_b$ memory cycles.[1]

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.

EXAMPLE

Zero the field starting at location TOTAL.

[1] Add one memory cycle to each of these times if the instruction is being executed in a Type 2201 processor.

EASYCODER
CODING FORM

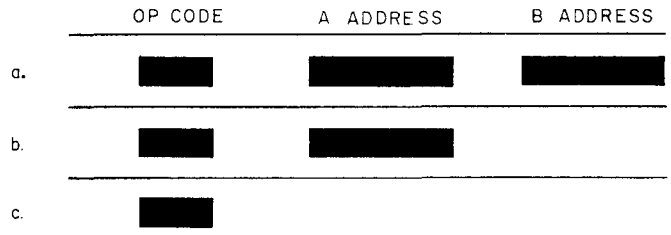
PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

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

NOTE: Information bits as well as zone bits are cleared to zero by this operation.

ZA | ZERO AND ADD | FEATURES 010 & 011

FORMAT



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). 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 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.

TIMING

Formats a, b, and c:

$T = N_i+1+N_w+N_b$ memory cycles. [1]

ADDRESS REGISTERS AFTER OPERATION

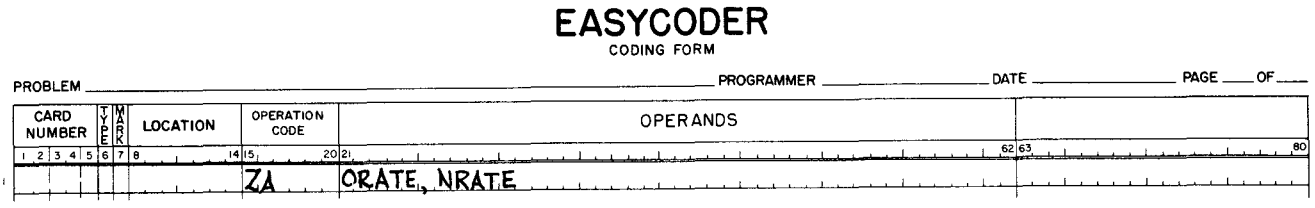
	SR	AAR	BAR
Format a:	NXT	$A-N_w$	$B-N_b$
Format b:	NXT	$A-N_a$	$A-N_a$
Format c:	NXT	A_p-N_w	B_p-N_b

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.

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.



ZS | ZERO AND SUBTRACT | FEATURES 010 & 011

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	[redacted]	[redacted]	[redacted]
b.	[redacted]	[redacted]	
c.	[redacted]		

[1] Add one memory cycle to this formula if the instruction is being executed in a Type 2201 processor.

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.

TIMING

Formats a, b, and c:

$T = N_i+1+N_w+N_b$ memory cycles.[1]

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.

[1] Add one memory cycle to this formula if the instruction is being executed in a Type 2201 processor.

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
				ZS	PROFIT

M MULTIPLY

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	[redacted]	[redacted]	[redacted]
b.	[redacted]	[redacted]	
c.	[redacted]		

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.

TIMING

Formats a, b, and c:

Types 201-1, 201-2, and 1201 processors:

$T = N_i+5+2N_a+2Z_{la}+5N_{mr}-Z_{mr}+s(N_a-Z_{ta})+2(N_a-Z_{ta})(N_{mr}-Z_{mr})$ memory cycles.

TYPE 2201 PROCESSOR:

$T = N_i+8+2N_a+2Z_{ta}+5N_{mr}-Z_{mr}+SUM(N_a-Z_{ta})+3(N_a-Z_{ta})(N_{mr}-Z_{mr})$ memory cycles.

Representative times for the Types 201-1, 201-2, 1201, and 2201 processors are given in note 7.

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 a standard feature on all processors but the Type 201, on which it is not available.

7. Listed below are representative multiply times (in microseconds) for the Type 201-1, 201-2, 1201, and 2201 processors. It is assumed that the

three-character addressing mode is used and that each multiplier digit has the median value of 4.5.

TYPE 201-1 AND 201-2 MULTIPLY TIMES (MICROSECONDS)						
		NO. OF CHARACTERS IN MULTIPLICAND				
		1	2	3	4	5
NUMBER OF CHARACTERS IN MULTIPLIER	1	51	68	85	102	119
	2	74	104	134	164	194
	3	97	140	183	226	269
	4	120	176	232	288	344
	5	143	212	281	350	419

TYPE 1201 MULTIPLY TIMES (MICROSECONDS)						
		NO. OF CHARACTERS IN MULTIPLICAND				
		1	2	3	4	5
NUMBER OF CHARACTERS IN MULTIPLIER	1	39.8	52.5	65.3	78	90.8
	2	57	79.5	102	124.5	147
	3	74.3	106.5	138.8	171	203.3
	4	91.5	133.5	175.5	217.5	259.5
	5	108.8	160.5	212.3	264	315.8

TYPE 2201 MULTIPLY TIMES (MICROSECONDS)						
		NO. OF CHARACTERS IN MULTIPLICAND				
		1	2	3	4	5
NUMBER OF CHARACTERS IN MULTIPLIER	1	28	36	44	52	60
	2	39	53	67	81	95
	3	50	70	90	110	130
	4	61	87	113	139	165
	5	72	104	136	168	200

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.

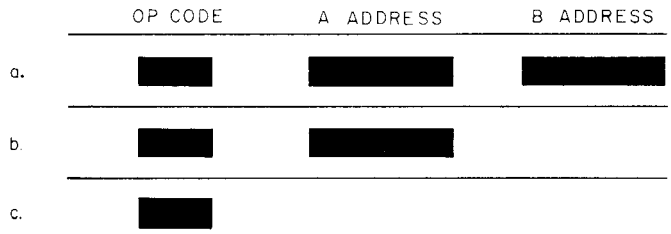
EASYCODER
CODING FORM

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 in the field 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 8-13).

Format b: The signed decimal integer in the field 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 8-13).

Format c: The signed decimal integer in the field 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 8-13).

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.

TIMING

Formats a, b, and c:

TYPES 201-1, 201-2, AND 1201 PROCESSORS:

$T = N_i+4+2N_a$ memory cycles if divisor = 0.

$T = N_i+17.5+4.5N_a+15.5Z_{1a}+12.5N_{dd}+15N_a(N_{dd}-N_a+Z_{1a})$ memory cycles if $(N_a-Z_{1a})(N_{dd})$ and divisor$\neq$0.

$T = N_i+7+4N_a$ memory cycles if $(N_a-Z_{1a}) > (N_{dd})$.

TYPE 2201 PROCESSOR:

$T = N_i+7+2N_a$ memory cycles if divisor = 0.

$T = N_i+9+2Z+5N_a+3Z_{1d}+N_q(15N_a-2Z_{1a}+18.25)$ memory cycles if $(N_a-Z_{1a}) \leq (N_{dd}-Z_{1d})$ and divisor$\neq$0.

$T = N_i+9+2N_a+2N_{dd}$ memory cycles if $N_a > N_{dd}$ and $(N_a-Z_{1a}) > (N_{dd}-Z_{1d})$.

$T = N_i+10+N_a+3N_{dd}$ memory cycles if $N_a \geq N_{dd}$ and $(N_a-Z_{1a}) > (N_{dd}-Z_{1d})$.

Representative divide times for the Type 201-1, 201-2, 1201, and 2201 processors are given in note 10.

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 sign 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 8-23).
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 a standard feature on all processors but the Type 201, on which it is not available.

9. Listed below are representative divide times (in microseconds) for the Type 201-1, 201-2, 1201, and 2201 processors. It is assumed that the processor is in the three-character addressing mode in all cases.

TYPE 201 DIVIDE TIMES (MICROSECONDS)						
		NUMBER OF CHARACTERS IN DIVIDEND				
		1	2	3	4	5
NUMBER OF CHARACTERS IN DIVISOR	1	83	138	193	248	303
	2	44	117	202	287	372
	3	52	52	151	266	381
	4	60	60	60	185	330
	5	68	68	68	68	219

TYPE 1201 DIVIDE TIMES (MICROSECONDS)						
		NUMBER OF CHARACTERS IN DIVIDEND				
		1	2	3	4	5
NUMBER OF CHARACTERS IN DIVISOR	1	62	103.5	145	186	227
	2	34.5	88	151.5	215	279
	3	40.5	40.5	94	199.5	286
	4	46.5	46.5	52.5	139	247.5
	5	52.5	52.5	52.5	52.5	164

TYPE 2201 DIVIDE TIMES (MICROSECONDS)						
		NUMBER OF CHARACTERS IN DIVIDEND				
		1	2	3	4	5
NUMBER OF CHARACTERS IN DIVISOR	1	54	88	121	154	187
	2	22	74	123	170	219
	3	24	26	94	158	221
	4	26	28	30	114	193
	5	28	30	32	34	134

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

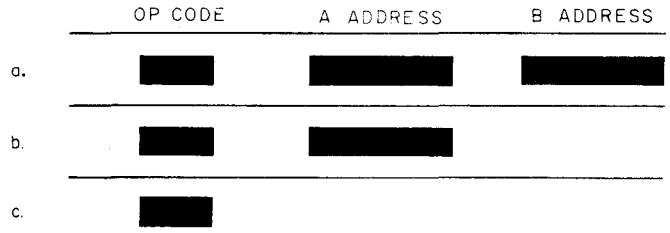
Units position of remainder ($B+N_{dd}-1$) = 1000+4-1 = location 1003

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.

TIMING

Formats a, b, and c:

$T = N_i + 1 + 3N_w$ memory cycles.[1]

[1] Add one memory cycle to this formula if the Extract instruction is being executed in a Type 2201 processor.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	$A\text{-}N_w$	$B\text{-}N_w$
Format b:	NXT	$A\text{-}N_w$	$B_p\text{-}N_w$
Format c:	NXT	$A_p\text{-}N_w$	$B_p\text{-}N_w$

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	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62 63 ... 80
				EXT	CON, BASE

HA	HALF ADD (Exclusive Or)

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	[redacted]	[redacted]	[redacted]
b.	[redacted]	[redacted]	
c.	[redacted]		

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

c.

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.

TIMING

Formats a, b, and c:

$T = N_i+1+3N_w$ memory cycles.[1]

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 data 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 data contents of SEVEN and TOO are identical.)

EASYCODER
CODING FORM

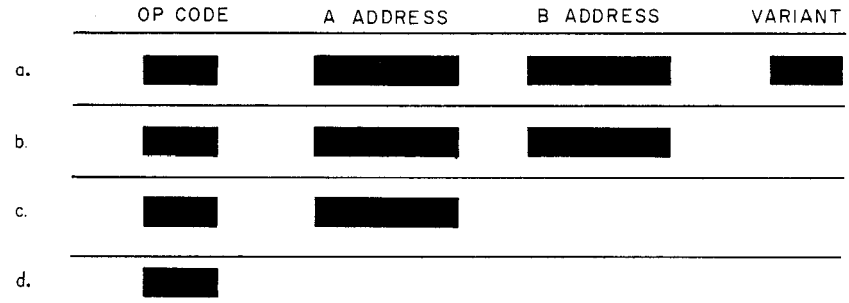
PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62
				HA	TOO, SEVEN

[1] Add one memory cycle to this formula if the Half Add instruction is being executed in a Type 2201 processor.

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.

TIMING

Formats a, b, c, and d:

$T = N_i + 4$ memory cycles.[1]

[1] Add one memory cycle to this formula if the Substitute instruction is being executed in a Type 2201 processor.

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
				SST	STET,STET+20,60	

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
				SST	256,656,17	

C | COMPARE

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	[redacted]	[redacted]	[redacted]
b.	[redacted]	[redacted]	
c.	[redacted]		

FUNCTION

Format a: The data in the B field is compared bit-by-bit to 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 to 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 to 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.

TIMING

Formats a, b, and c:

$T = N_i+2+N_w+N_b$ memory cycles.[1]

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	$A\text{-}N_w$	$B\text{-}N_b$
Format b:	NXT	$A\text{-}N_w$	$B_p\text{-}N_b$
Format c:	NXT	$A_p\text{-}N_w$	$B_p\text{-}N_b$

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 on page 8-40.

[1] Add two memory cycles to this formula if the instruction is executed in a Type 2201 processor.

COMPARISON RESULT	BRANCH CONDITION
B<A	Low Compare
B=A	Equal Compare
B≤A	Low or Equal Compare
B>A	High Compare
B≠A	Unequal Compare
B≥A	High or Equal Compare

EXAMPLE

Compare Item Number to 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
[redacted]	[redacted]		

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.

TIMING

$T = N_i + 2$ memory cycles.[1]

[1] Add one memory cycle to this formula if the Branch instruction is being executed in a Type 2201 processor.

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR
JI (A)	A	NXT

NOTE

The address bits of the A address are placed in AAR during the extraction of this instruction. 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.

EXAMPLE

Select the next instruction from the location tagged SUB6.

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
				B	SUB6

BCT BRANCH ON CONDITION TEST

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.	[redacted]	[redacted]		[redacted]
b.	[redacted]			

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.

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
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.

WORD MARKS

Formats a and b:

Word marks are not affected by this instruction.

TIMING

Formats a and b:

$T = N_i + 2$ memory cycles.[1]

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. stored (and cleared) by the Store Variant and Indicators instruction;

   c. restored by the Restore Variant and Indicators instruction;

   d. restored by the Resume Normal Mode instruction; and

   e. stored when an external interrupt occurs.

3. The address bits of the A address (if any) are placed in AAR during the extraction of this instruction. 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.

[1] Add two memory cycles to this formula if the instruction is executed in a Type 2201 processor.

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 processor and are not available with the Model 200 or 1200 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.

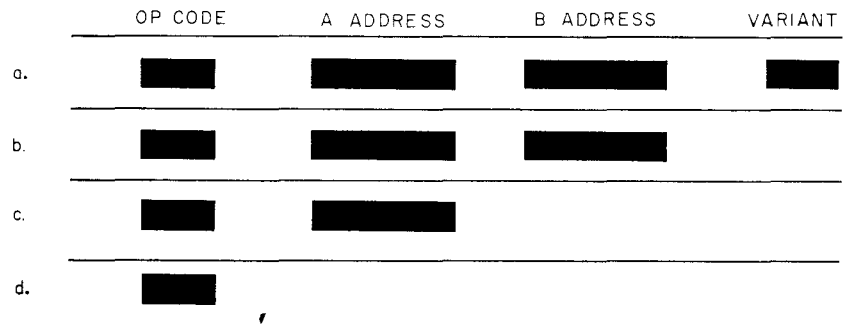
EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1					S	CREDIT, TOTAL
2					BCT	BZRO, 60

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.

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 and the word-mark bit is 1.
16	The character at B contains a negative sign and 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 and the item-mark bit is 1.
26	The character at B contains a negative sign and 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 and the B bit is 1.
36	The character at B contains a record mark and a negative sign.

Series 200 processors which are equipped with Feature 010 or 011 (see Figure 1-5) 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 8-11. BCC Test Conditions with Advanced Programming Feature

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.
6X	The character at B contains an item mark.
7X	The character at B contains a word mark. (This is a special case; see note).

NOTE: 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. However, if the rightmost digit is 0 and and the leftmost digit is 7, the branch is an unconditional branch.

If both octal digits specify conditions, the branch occurs if both conditions are met. However, if the leftmost digit is 7, the branch occurs if either the condition specified by the rightmost digit is met or the character at B contains a word mark.

WORD MARKS

Formats a, b, c, and d:

Word marks are not affected by this instruction.

TIMING

Formats a, b, c, and d:

$T = N_i + 4$ memory cycles.[1]

ADDRESS REGISTERS AFTER OPERATION

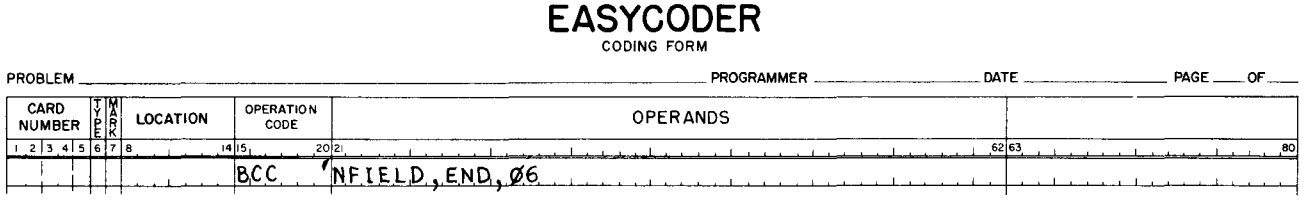
	SR	AAR	BAR	
Format a:	JI (A)	A	NXT	BRANCH
	NXT	A	B-1	NO BRANCH
Format b:	JI (A)	A	NXT	BRANCH
	NXT	A	B-1	NO BRANCH
Format c:	JI (A)	A	NXT	BRANCH
	NXT	A	B_p-1	NO BRANCH
Format d:	JI (A_p)	A_p	NXT	BRANCH
	NXT	A_p	B_p-1	NO BRANCH

NOTES

1. If the octal configuration of the variant character is 00, or 70, the branch is unconditional.
2. The address bits of the A address (if any) are placed in AAR during the extraction of the BCC instruction. 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 Types 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

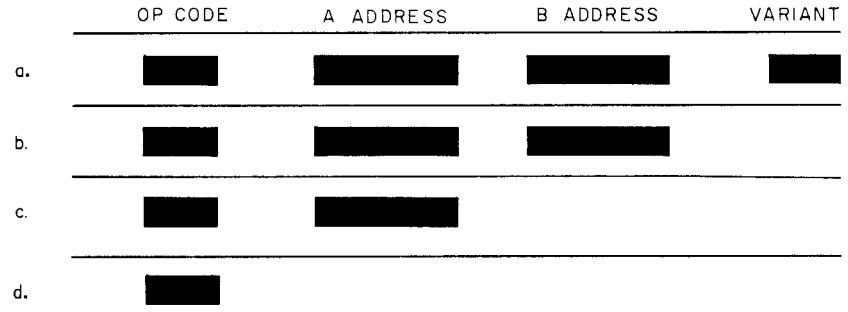
PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 14	15 20	21 62 63 80
				BCC	NFIELD,END,Ø6

[1] Add one memory cycle to this formula if the instruction is being executed in a Type 2201 processor.

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.

TIMING

Formats a, b, c, and d:

$T = N_i + 4$ memory cycles.[1]

[1] Add one memory cycle to this formula if the instruction is being executed in a Type 2201 processor.

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 address bits of the A address (if any) are placed in AAR during the extraction of the BCE instruction. 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.

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	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+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.

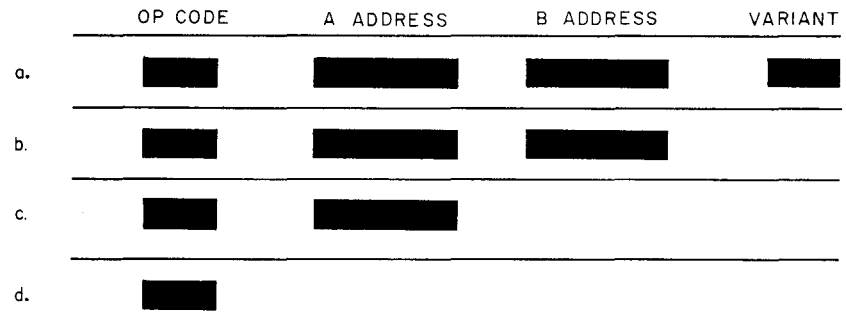
EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ___ OF ___

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5 6 7 8 ... 14	15 ... 20	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 SR1 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.

TIMING

Formats a, b, c, and d:

$T = N_i + 4$ memory cycles.[1]

ADDRESS REGISTERS AFTER OPERATION

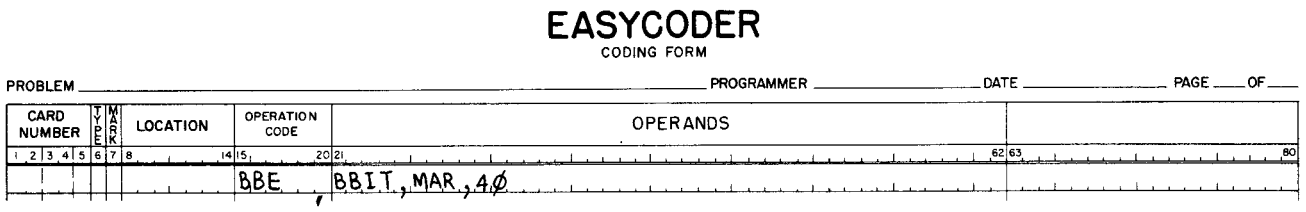
	SR	AAR	BAR	
Format a:	JI (A)	A	NXT	BRANCH
	NXT	A	B-1	NO BRANCH
Format b:	JI (A)	A	NXT	BRANCH
	NXT	A	B-1	NO BRANCH
Format c:	JI (A)	A	NXT	BRANCH
	NXT	A	B_p-1	NO BRANCH
Format d:	JI (A_p)	A_p	NXT	BRANCH
	NXT	A_p	B_p-1	NO BRANCH

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 address bits of the A address (if present) are placed in AAR during the extraction of the instruction. 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.

EXAMPLE

Branch to the location tagged BBIT if the character at the location tagged MAR contains a "1" in the B-bit position. Otherwise, continue the program in sequence.



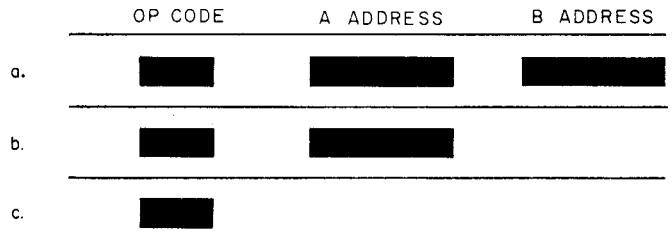
[1] Add one memory cycle to this formula if the instruction is being executed in a Type 2201 processor.

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

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.

TIMING

Formats a, b, and c:

$T = N_i+3$ memory cycles.[1]

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

[1] Add one memory cycle to this formula if the instruction is being executed in a Type 2201 processor. Subtract one memory cycle from this formula if the instruction is being executed in a Type 1201 processor in format a.

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	WAR	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62
				SW	435

SI	SET ITEM MARK

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	[redacted]	[redacted]	[redacted]
b.	[redacted]	[redacted]	
c.	[redacted]		

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.

TIMING

Formats a, b, and c:

$T = N_i + 3$ memory cycles.[1]

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	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62 63 ... 80
				ST	ENT, ENT+80

CW | CLEAR WORD MARK

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	[redacted]	[redacted]	[redacted]
b.	[redacted]	[redacted]	
c.	[redacted]		

[1] Add one memory cycle to this formula if the instruction is being executed in a Type 2201 processor. Subtract one memory cycle from this formula if the instruction is being executed in a Type 1201 processor in format a.

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.

TIMING

Formats a, b, and c:

$T = N_i + 3$ memory cycles.[1]

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

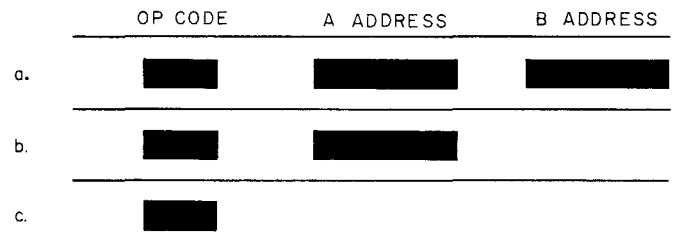
PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62	63 ... 80
				CW	400,435	

[1] Add one memory cycle to this formula if the instruction is being executed in a Type 2201 processor.

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.

TIMING

Formats a, b, and c:

$T = N_i+3$ memory cycles.[1]

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.

[1] Add one memory cycle to this formula if the instruction is being executed in a Type 2201 processor.

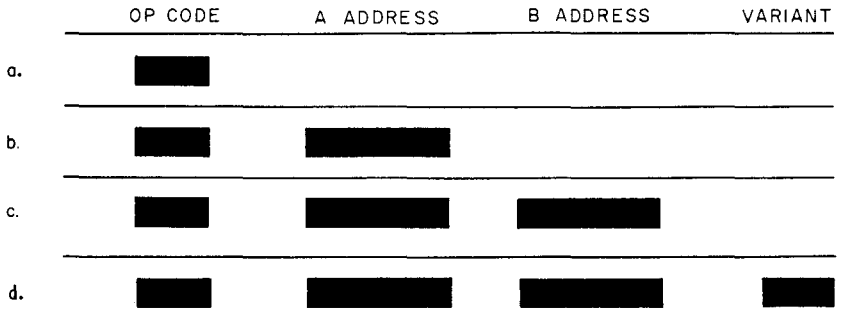
EASYCODER
CODING FORM

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	
2						

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.

TIMING

Formats a, b, c, and d:

$T = N_i + 2$ memory cycles.[1]

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 or 2201 central processor is equipped with the Storage 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	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 ... 14	15 ... 20	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	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62	63 ... 80
				H	9,9	

[1] Add two memory cycles to this formula if the instruction is being executed in a Type 2201 processor.

NOP	NO OPERATION

FORMAT

OP CODE	A ADDRESS	B ADDRESS

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.

TIMING

T = 3 memory cycles.[1]

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.

EXAMPLE

Reserve one storage location for an operation code such as Branch (B). When the op code B is inserted, the NOP instruction will be modified to branch to location SWX.

EASYCODER
CODING FORM

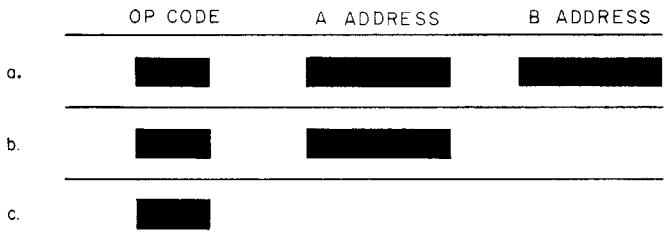
PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62
				NOP	SWX

[1] Add one memory cycle to this formula if the instruction is being executed in a Type 2201 processor. Subtract one cycle from the formula if the instruction is executed in a Type 1201 processor.

MCW	MOVE CHARACTERS TO WORD MARK

FORMAT



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 is required in the shorter of the two fields. The operation terminates when this word mark is sensed.

TIMING

Formats a, b, and c:

$T = N_i+1+2N_w$ memory cycles.[1]

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.

[1] Add one memory cycle to this formula if the instruction is being executed in a Type 2201 processor.

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	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1					MCW	187,825	
2					MCW	180	
3					MCW	168	
4					MCW	155	
5							

LCA — LOAD CHARACTERS TO A-FIELD WORD MARK

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	[redacted]	[redacted]	[redacted]
b.	[redacted]	[redacted]	
c.	[redacted]		

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. The operation terminates when this word mark is transferred to the B field.

TIMING

Formats a, b, and c:

$T = N_i+1+2N_a$ memory cycles.[1]

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	$A-N_a$	$B-N_a$
Format b:	NXT	$A-N_a$	B_p-N_a
Format c:	NXT	A_p-N_a	B_p-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. A record mark appearing in the A field terminates the operation.
3. 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.
4. 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

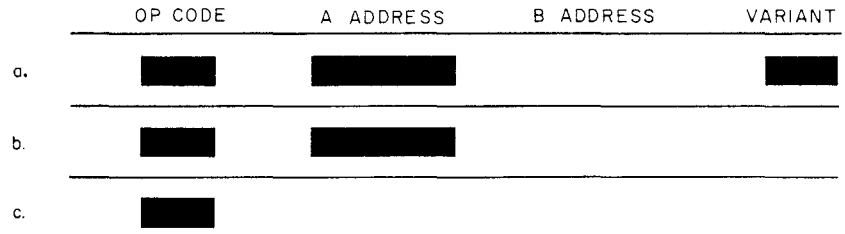
PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

	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							

[1] Add one memory cycle to this formula if the instruction is executed in a Type 2201 processor.

SCR STORE CONTROL REGISTERS

FORMAT



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.

Table 8-12. Control Register Contents Stored by SCR Instruction

Addressing Mode	Amount of Control Register Stored
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 18 bits (three characters) of the control register.
NOTE: All bit positions not required to address the largest memory address in a user's system are set to zeros in the A field.	

The valid variant characters and the control register each character represents are listed in Table 8-13.

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
01	CLC1	21	CLC4
02	CLC2	22	CLC5
03	CLC3	23	CLC6
05	CLC1'	25	CLC4'
11	SLC1	31	SLC4
12	SLC2	32	SLC5
13	SLC3	33	SLC6
15	SLC1'	35	SLC4'
64	CSR	70	BAR
66	EIR	76	IIR
67 (see note 2)	AAR	77	SR

WORD MARKS

Formats a, b, and c:

A-operand punctuation neither affects nor is affected by this instruction.

TIMING

Formats a, b, and c:

T = N_i+5 memory cycles.[1]

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), AAR 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.

[1] Add two memory cycles to this formula if the instruction is executed in a Type 2201 processor.

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	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62	63 ... 80
			SUB	SCR	EXIT+3,7Ø	
			EXIT	B	Ø	

LCR | LOAD CONTROL REGISTERS | FEATURE 011

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.	[redacted]	[redacted]		[redacted]
b.	[redacted]	[redacted]		
c.	[redacted]			

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	18-bit address is loaded into the register.

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 or is affected by this instruction.

TIMING

Formats a, b, and c:

$T = N_i + 5$ memory cycles.[1]

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 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.

[1] Add two memory cycles to this formula if the instruction is being executed in a Type 2201 processor.

3. The LCR op code is a "privileged" op code which has special significance when used with a Type 1201 or 2201 processor equipped with the Storage Protect Feature (see Appendix 2.)

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

FORMAT

	OP CODE	A ADDRESS	B ADDRESS	VARIANT
a.	[redacted]			[redacted]
b.	[redacted]			

FUNCTION

Format a: The Change Addressing Mode instruction is used to specify the following conditions, as designated by the variant character:

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.)

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

WORD MARKS

Formats a and b:

Word marks are not affected by this instruction.

TIMING

Formats a and b:

T = N_i+2 memory cycles.[1]

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 7-9 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 1).

   The larger addresses are necessary to address larger continuous portions of memory. For instance, a two-character address can address 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 $262,144_{10}$).

3. 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

[1] Subtract one memory cycle if the instruction is being executed in a Type 1201 processor. Add one cycle if the instruction is executed in a Type 2201 processor.

(see page 8-72), 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 code is executed as a CSM op code.

4. 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.)

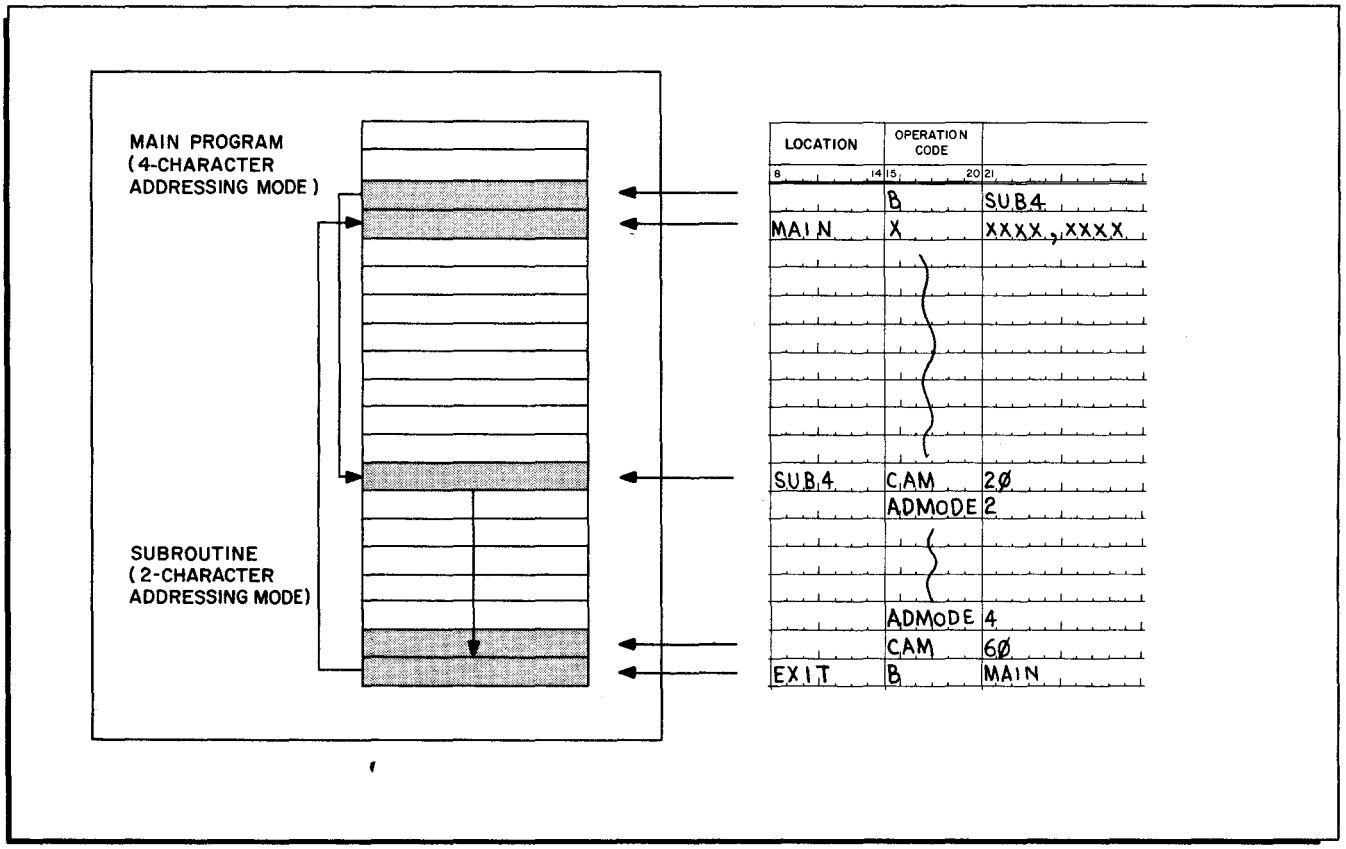
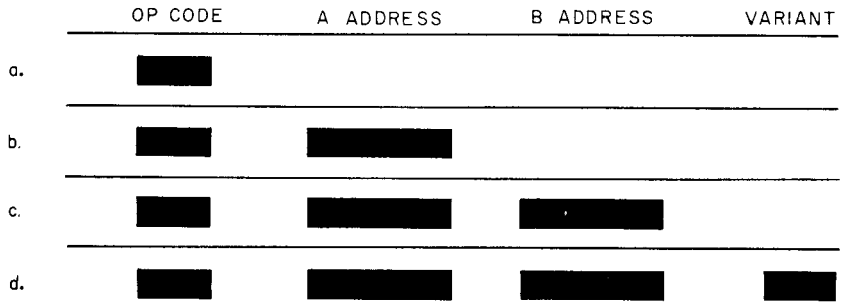


Figure 8-5. Changing Addressing Modes via CAM Instruction

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.

TIMING

Formats a, b, c, and d:

$T = N_i + 3$ memory cycles.[1]

[1] Subtract one memory cycle from this formula if the instruction is being executed in a Type 1201 processor. Add one cycle if the instruction is executed in a Type 2201 processor.

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 8-67) can be used to specify the contents of CSR.
2. When the "trap" mode of instruction execution is specified by the Change Addressing Mode instruction (see page 8-69), 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.
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.

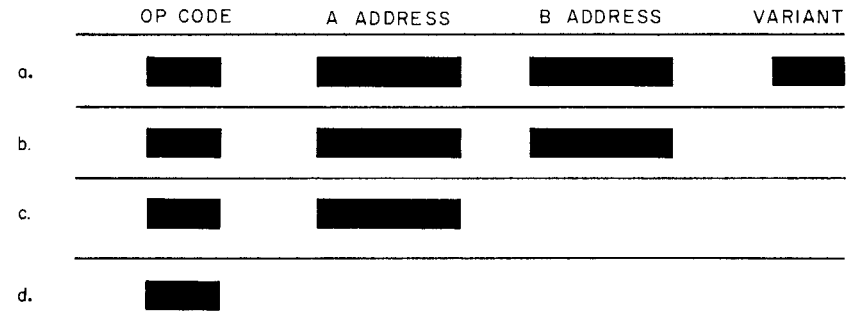
EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

	CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
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 specified 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 position 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.

Table 8-16 (cont). Extended Move Conditions

Variant Character (Octal)	Condition
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.
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.

TIMING

Formats a, b, c, and d:

$T = N_i+1+2N_a$ memory cycles.[1]

[1] Add one memory cycle to this formula if the instruction is being executed in a Type 2201 processor.

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-marks 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.

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
1				EXM	RESV,WORK,41
2					

MAT | MOVE AND TRANSLATE | FEATURES 010 & 011

FORMAT

OP CODE	A ADDRESS	B ADDRESS	VARIANT 1	VARIANT 2
[redacted]	[redacted]	[redacted]	[redacted]	[redacted]

FUNCTION

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.

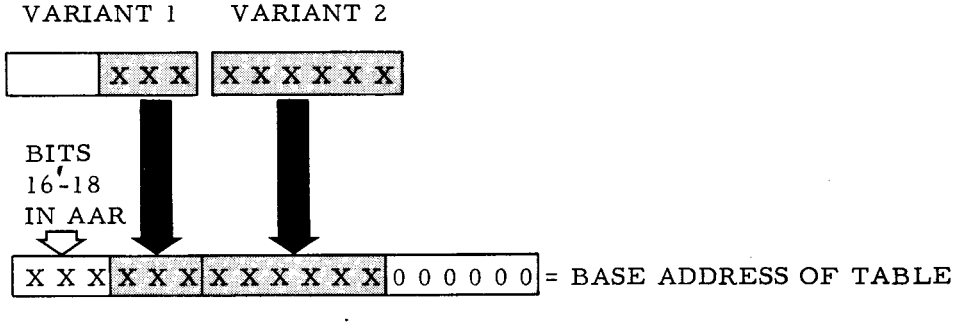
The A address specifies the location of the rightmost character to be translated. The B address specifies the location into which the translated equivalent of the rightmost A-field character will be moved.

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.

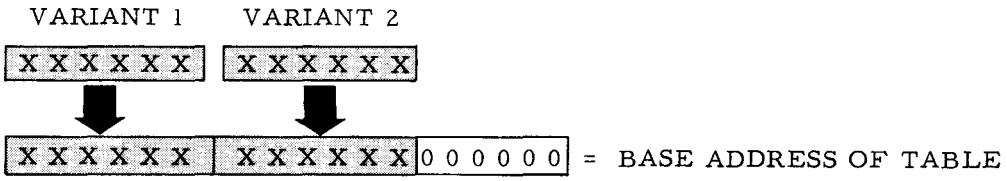
The address within the translation table which contains the translated equivalent of an A-field character is formed by combining the A-field character with the two variant characters. The method of combining these three characters depends on the addressing mode being used, as described below.

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, and 18) are taken from the contents of the A-address register (AAR). If the processor is in the four-character addressing mode (see next page), the entire six-bit contents of variant 1 form the leftmost six bits of the base address.

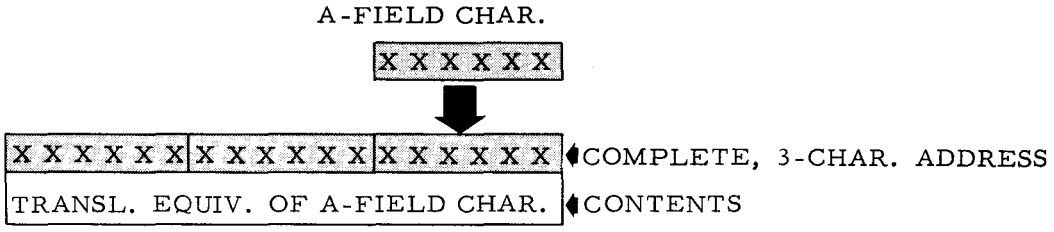
Two- or Three-Character Addressing Mode



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, 18-bit address. This complete address contains the translated equivalent of the appended A-field character character (see below).



Note that because of the positions of variant 1 and variant 2 in the total three-character 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 6).

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.

WORD MARKS

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.

TIMING

$T = N_i + 3N_a$ memory cycles. [1]

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR
NXT	A-N_{ct}	B-N_{ct}

[1] Add four memory cycles to this formula if the instruction is being executed in a Type 2201 processor.

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 programmer can use a symbolic tag in place of the variant characters of this instruction by previously equating the variant characters to the tag via a CEQU assembly control statement (see page 7-11).
6. 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 7-7) containing an operand of 64.

EXAMPLE

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). 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

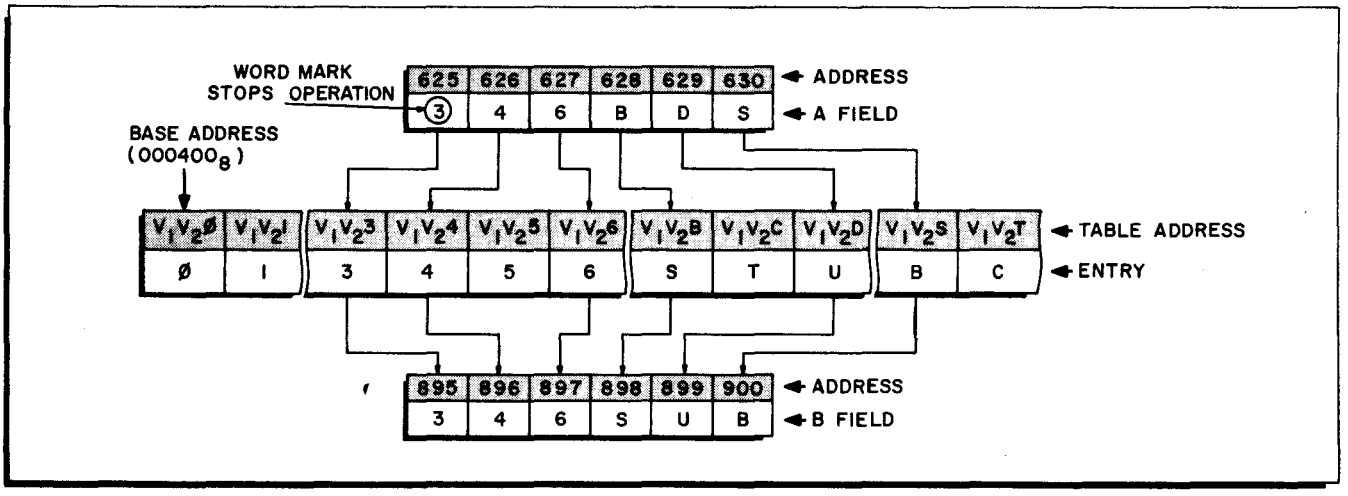
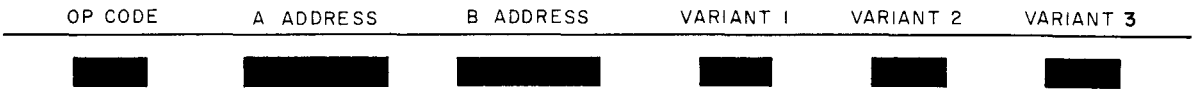


Figure 8-6. MAT Operation

MIT | MOVE ITEM AND TRANSLATE

FORMAT



FUNCTION

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 translated 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.

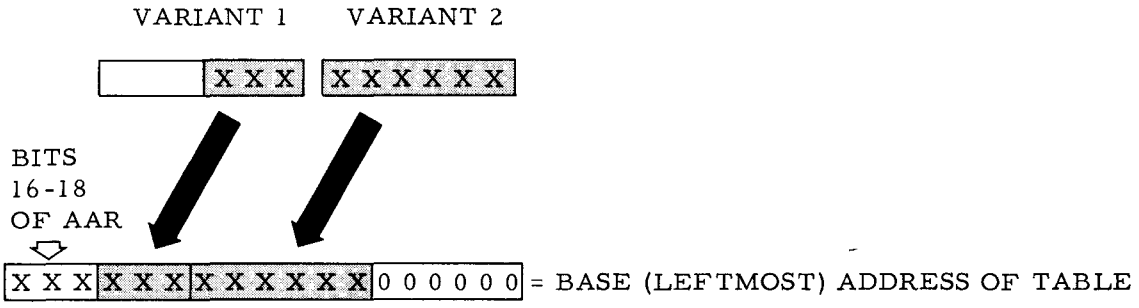
Table 8-17. Size of Information Units in MIT Operation

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.

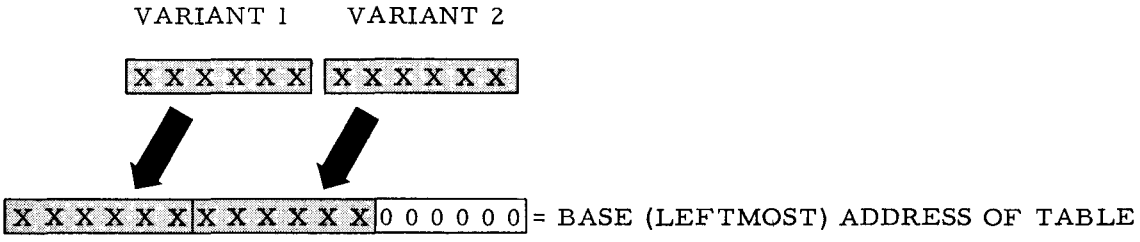
The desired equivalent of an A-item information unit is taken from the stored translation table and moved to the B item. Thus, if the desired equivalent is a six-bit character, each table entry occupies one six-bit character location in the table. If the desired equivalent is a 12-bit character, each 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 the leftmost six bits of the base address.

Two- or Three-Character Addressing Mode



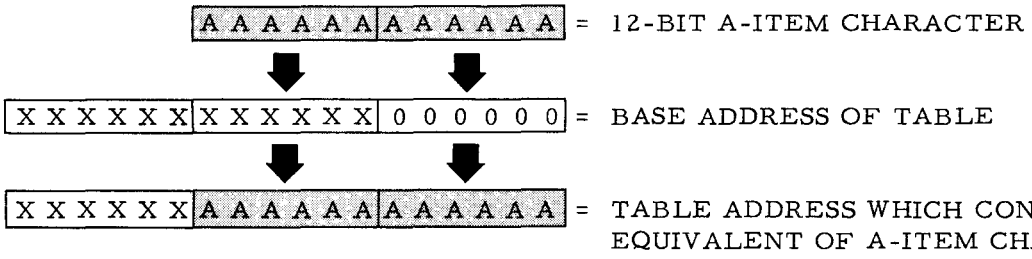
Four-Character Addressing Mode



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]

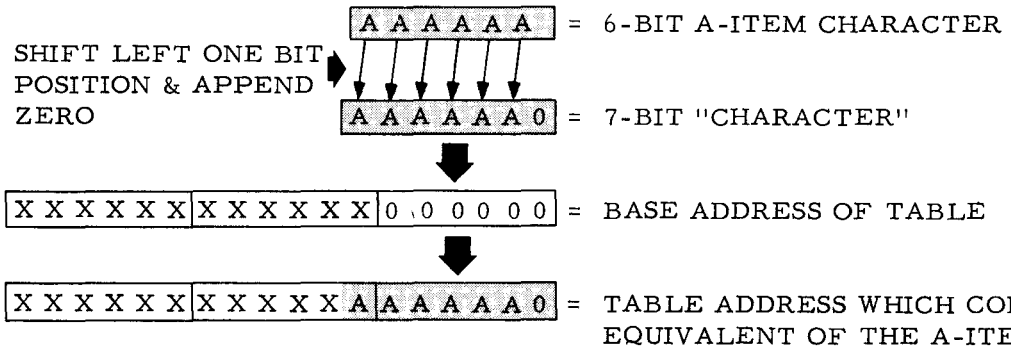
If each table entry is a six-bit character (variant 3 = 00 or 01), the 6- or 12-bit A-item character is superimposed over the rightmost bit positions of the base address. The illustration below shows a 12-bit A-item character being superimposed over the base address, where A = an A-item bit and X = a base address bit.

[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.



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.



PUNCTUATION MARKS

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 6).

TIMING

$T = N_i + N_a + 2(N_{ut})(NB_u)$ memory cycles.[1]

[1] Add two memory cycles to this formula if the instruction is being executed in a Type 1201 processor. Add four memory cycles if it is executed in a Type 2201 processor.

ADDRESS REGISTERS AFTER OPERATION

SR	CSR	AAR	BAR	
NXT	CSR_p	$A+(NA_u)(N_{ut})$	$B+(NB_u)(N_{ut})$	A-ITEM ITEM MARK 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. The programmer can use a symbolic tag in place of the variant characters of this instruction by previously equating the variant characters to the tag via a CEQU assembly control statement (see page 7-11).
6. 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 8-90) and subsequently tested for identification in the routine.
7. The base address of the translation table must always be a multiple of 64 due to the positions of variants 1 and 2 in the total three-character address. This is compatible with the translation requirements of six-bit characters. However, if information units greater than six bits in length are involved in the translation, a larger table may be required. For instance, if a character were to be translated into a seven-bit information unit and each seven-bit equivalent were unique, a 128-character table would be required. The MORG assembly control statement (see page 7-7) can be used to assign memory locations to the table starting with the next available memory location whose address is a multiple of 64, 128, 256, etc., whichever multiple is necessary to allocate the correct number of locations.
8. 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 =
Variant 2:	10 = the address of table (location 512)
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

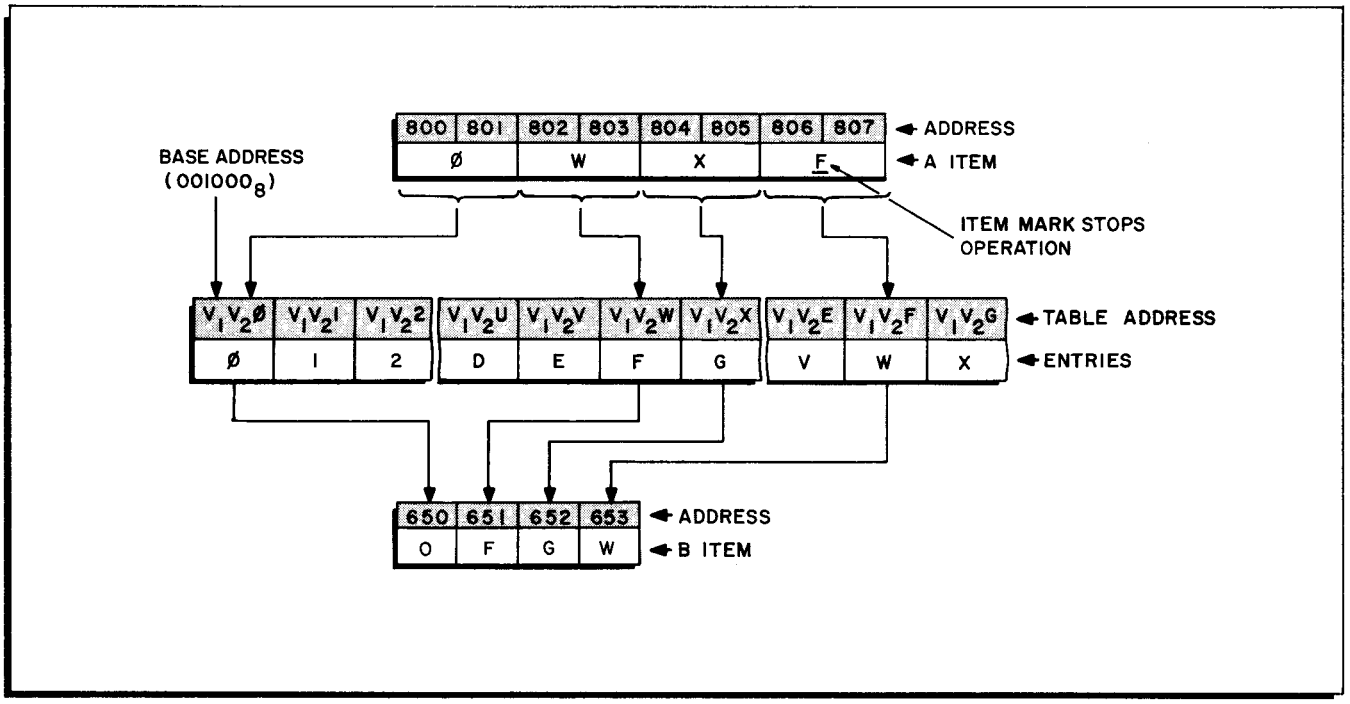


Figure 8-7. MIT Operation

LIB	LOAD INDEX/BARRICADE INDICATOR	FEATURE 1114 & 1117

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	[redacted]	[redacted]	
b.	[redacted]		

FUNCTION

Format a: The single character 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 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.) The protected memory area is defined by the single-character contents of the A address as shown in Table 8-18.

Format b: The single character specified by the contents of the A-address register (AAR) is loaded into the index/barricade register (IBR), specifying the number of a 4,096-character memory bank. 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.) The protected memory area is defined by the contents of the previous A address as shown in Table 8-18.

Table 8-18. Leftmost Boundaries of Protected Memory

Contents of A Address		Leftmost Boundary of Protected Memory	Contents of A Address		Leftmost Boundary of Protected Memory
Octal	Decimal		Octal	Decimal	
0	0	0	40	32	131,072
1	1	4,096	41	33	135,168
2	2	8,192	42	34	139,264
3	3	12,288	43	35	143,360
4	4	16,384	44	36	147,456
5	5	20,480	45	37	151,552
6	6	24,576	46	38	155,648
7	7	28,672	47	39	159,744
10	8	32,768	50	40	163,840
11	9	36,864	51	41	167,936
12	10	40,960	52	42	172,032
13	11	45,056	53	43	176,128
14	12	49,152	54	44	180,224
15	13	53,248	55	45	184,320
16	14	57,344	56	46	188,416
17	15	61,440	57	47	192,512
20	16	65,536	60	48	196,608
21	17	69,932	61	49	200,704
22	18	73,728	62	50	204,800
23	19	77,824	63	51	208,896
24	20	81,920	64	52	212,992
25	21	86,016	65	53	217,088
26	22	90,112	66	54	221,184
27	23	94,208	67	55	225,280
30	24	98,304	70	56	229,376
31	25	102,400	71	57	233,472
32	26	106,496	72	58	237,568
33	27	110,592	73	59	241,664
34	28	114,688	74	60	245,760
35	29	118,784	75	61	249,856
36	30	122,880	76	62	253,952
37	31	126,976	77	63	258,048

WORD MARKS

Formats a and b:

Word marks are not affected by this instruction.

TIMING

Formats a and b:

T = N_i+3 memory cycles.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	A	B_p
Format b:	NXT	A_p	B_p

NOTES

1. The 15 additional index registers which are included in the Storage Protect Feature are located in the leftmost 60 character locations of the main memory bank specified by this instruction. These locations can be used as normal storage locations when they are not 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 or 2201 processor (see Appendix E.)

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	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
	1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62	63 ... 80
1					LIB	MP2	
2							
3							

SIB STORE INDEX/BARRICADE INDICATOR — FEATURE 1114 & 1117

FORMAT

	OP CODE	A ADDRESS	B ADDRESS
a.	[redacted]	[redacted]	
b.	[redacted]		

FUNCTION

Format a: The single-character contents of the index/barricade register (IB) are stored in the character location specified by the A address. All high-order bit positions in A which are not used to specify the contents of the index/barricade indicator are cleared to zeros.

Format b: The single-character contents of the index/barricade register (IB) are stored in the character location specified by the contents of the A-address register (AAR). All high-order bit positions in A which are not used to specify the contents of the index/barricade indicator are cleared to zeros.

WORD MARKS

Formats a and b:

Word marks are not affected by this instruction.

TIMING

Formats a and b:

$T = N_i + 3$ memory cycles.

ADDRESS REGISTERS AFTER OPERATION

	SR	AAR	BAR
Format a:	NXT	A	B_p
Format b:	NXT	A_p	B_p

EXAMPLE

Store the contents of the index/barricade register in the single character location tagged PROT.

EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

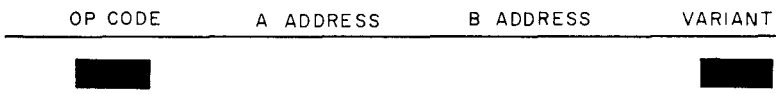
CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 ... 14	15 ... 20	21 ... 62	63 ... 80
				SIB	PROT	

INTERRUPT CONTROL

- STORE VARIANT AND INDICATORS
- RESTORE VARIANT AND INDICATORS
- MONITOR CALL
- RESUME NORMAL MODE

SVI STORE VARIANT AND INDICATORS

FORMAT



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 8-19 and encoding the resulting bit configuration as two octal digits.

Table 8-19. 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). These contents are identical to those described for V_2, above. 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). These indicators are cleared when their contents have been stored.
X 1 X X X X	The settings of the protect and proceed indicators and (if the processor is in the external interrupt mode) the setting of the internal interrupt (II) mode indicator.[1]

Table 8-19 (cont). Information Stored by SVI Instruction

VARIANT CHARACTER $V_6V_5V_4V_3V_2V_1$	INFORMATION STORED
X 1 X X X X (cont)	The protect and proceed indicators are cleared when their contents are stored.
1 X X X X X	The settings of the interrupt source indicators. The stored settings of these 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 8-95). 3. Whether an external interrupt (EI) address violation[2] has occurred (if the processor is in the external interrupt mode). 4. Whether an op code violation[2] has occurred (if the processor is in the internal interrupt mode). 5. Whether an internal interrupt (II) address violation[2] has occurred (if the processor is in the internal interrupt mode). The indicators referred to in 3 through 5, above, as well as the indicator which identifies the control panel (or console) as the interrupt source are cleared when their contents are stored.

[1] These indicators are included in a Type 1201 or 2201 processor equipped with the Storage Protect Feature (see Appendix D).

[2] EI address violation, op code violation, and II address violation are associated with the Storage Protect Feature (see Appendix D).

WORD MARKS

Word marks in the locations in which information is stored neither affect nor are affected by this operation. Program operation resumes with the first word-marked location following the stored information (the next sequential op code).

TIMING

$T = N_i+1+N_s+N_j$ memory cycles.[1]

[1] Add one memory cycle to this formula if the instruction is executed in a Type 1201 processor. Add two cycles if the instruction is executed in a Type 2201 processor.

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR
NXT	A_p	B_p

NOTES

1. Only the number of characters specified by the variant character are stored. They are stored in the order listed in Table 8-19: the contents of the variant register (if specified) are stored in the location immediately following the SVI instruction, etc., using only those locations actually required to store the requested information.

VARIANT BIT	STORED CHARACTER LOCATION BITS						
	I/M BIT	B BIT	A BIT	8 BIT	4 BIT	2 BIT	1 BIT
V_1	0 Item-mark	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 in V_2, above)			*	*	*	*
V_4	0	MPO: * 1=yes; 0=no.	DVC: * 1=yes; 0=no.	EXO: 1=yes; 0=no.	0	0	0
V_5	0	Protect indicator: 1=on; 2=off.	0	0	Proceed indicator: 1=on; 0=off. *	0	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. *	0	0	0	II Mode indicator: 1=on; 0=off.

* = Indicators that are cleared when their contents are stored.

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 following 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 or 2201 processor equipped with the Storage Protect Feature (see Appendix E).

7. This instruction is a standard feature on all processors but the Types 201 and 201-1, on which it is not available.

8. The method of coding interrupt service routines is described in Appendix D, "Interrupt Processing."

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.

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	SVI	45
2					

RVI | RESTORE VARIANT AND INDICATORS

FORMAT

OP CODE	A ADDRESS	B ADDRESS	VARIANT
[redacted]	[redacted]		[redacted]

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 8-20 and encoding the resulting bit configurations as two octal digits.

Table 8-20. 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). This information is identical to that described for V_2, above.
0 X 1 X X X	The settings of the indicators associated with the scientific unit (see Appendix F).
0 1 X X X X	The settings of the protect and proceed 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 or 2201 processor equipped with the Storage Protect Feature (see Appendix E).

WORD MARKS

Word marks neither affect nor are affected by this instruction.

TIMING

$T = N_i+2+N_r$ memory cycles.[1]

[1] Subtract one memory cycle from this formula if the instruction is executed in a Type 1201 processor. Add one cycle if the instruction is executed in a Type 2201 processor.

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR
NXT	A_p	B_p

NOTES

1. Each entry in the righthand column of Table 8-20 is stored in a single character location. Only the number of characters corresponding to the selected table entries are restored by the RVI instruction.
2. The RVI op code is a "privileged" op code that has special significance when used with a Type 1201 or 2201 processor equipped with the Storage Protect Feature (see Appendix E).
3. All information stored by an SVI instruction should be restored by the RVI instruction.
4. The format in which information is stored by an SVI instruction is shown in the table on page 8-92. 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."

EXAMPLE

Restore the contents of the variant register and auxiliary indicators register (AIR) that were previously stored by the SVI instruction example on page 8-93.

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.

TIMING

$T = N_i+2$ memory cycles.[1]

ADDRESS REGISTERS AFTER OPERATION

SR	EIR	AAR	BAR
JI (contents of EIR)	NXT	A_p	B_p

NOTES

1. This instruction must not be issued in the external interrupt mode.
2. The interrupt source indicators can be stored via an SVI instruction (see page 8-90). Their stored contents can then be interrogated by programmed instruction to determine the interrupt source.

EXAMPLE

Interrupt the central processor and branch to MONITOR, the location of the monitor program. The address tagged MONITOR was previously stored in EIR.

[1] Subtract one memory cycle to this formula if the instruction is executed in a Type 1201 processor. Add one cycle if the instruction is executed in a Type 2201 processor.

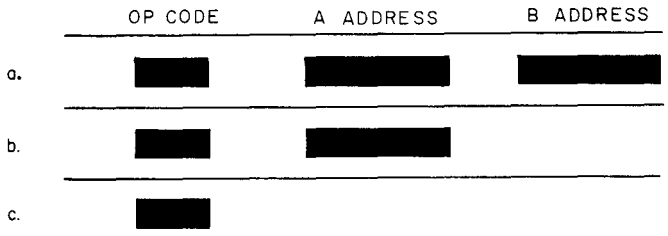
EASYCODER
CODING FORM

PROBLEM _____ PROGRAMMER _____ DATE _____ PAGE ____ OF ____

CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS
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.

TIMING

Formats a, b, and c:

$T = N_i + 3$ memory cycles.[1]

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 a "privileged" op code which has special significance when used with a Type 1201 or 2201 processor equipped with the Storage Protect Feature (see Appendix E).

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 ___

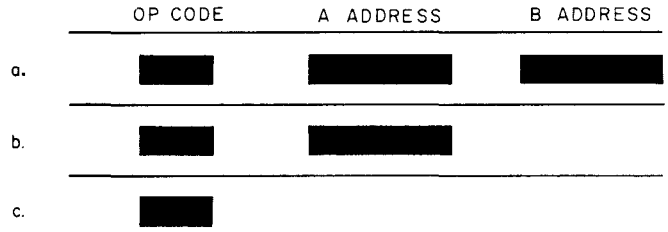
CARD NUMBER	TYPE	MARK	LOCATION	OPERATION CODE	OPERANDS	
1 2 3 4 5	6	7	8 14	15 20	21 62	63 80
1			RESUME	RNM		
2			EXT2	SVI	45	
3						
4					INTERRUPT ROUTINE	
5						
6						
7				B	RESUME	
8						

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 8-21. 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 8-21. 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.

TIMING

Formats a, b, and c:

$T = N_i+1+N_a+2N_b+2Z+2\$$ memory cycles.[1]

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.

EXAMPLES[2]

Data Field (A Field)	(0)000099$\bar{9}$
Control Word (B Field)	(b) bb, bb0. bb&&$\bar{0}$
Result of Edit	.99 -

Example 1.

Data Field (A Field)	(2) 5454986
Control Word (B Field)	(b) bb&bb&bbb
Result of Edit	254 54 986

Example 2.

[1] Add one memory cycle to this formula if the instruction is being executed in a Type 2201 processor.

[2] 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)	(0) 00450†
Control Word (B Field)	($) b, bb0. bb&CR*
Result of Edit	$ 4. 50 *

Example 3.

Data Field (A Field)	(0) 0897445
Control Word (B Field)	(b) bbb, b$0. bb
Result of Edit	$8, 974. 45

Example 4.

Data Field (A Field)	(0) 010450
Control Word (B Field)	(b) 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.

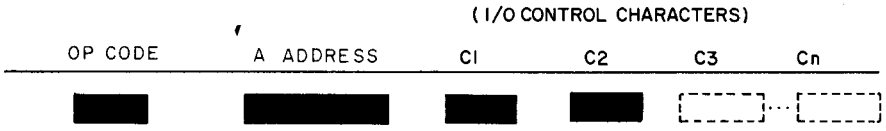
The PCB instruction performs two distinct functions: (1) it initiates strictly mechanical operations such as magnetic tape rewinds and card rejections; and (2) it causes a program branch to be performed contingent upon the setting of peripheral condition indicators. The latter facility allows programmed tests for such peripheral conditions as read or write errors, busy peripheral devices or control units, and magnetic tape unit at end of tape.

Detailed programming and operating information for Series 200 peripheral devices is provided in separate manuals and information bulletins. 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 publications.

PDT instructions are described starting on this page — first for all Series 200 devices except the Type 286 Multi-Channel Communication Control, and secondly for the Type 286. PCB instructions for all Series 200 devices except the Type 286 are described starting on page 8-117; the description of the Type 286 follows on page 8-131. 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.

PDT | PERIPHERAL DATA TRANSFER For: PUNCHED CARD EQUIPMENT
PAPER TAPE EQUIPMENT
PRINTER
MAGNETIC TAPE EQUIPMENT
RANDOM ACCESS DRUM
MASS MEMORY FILE
CONSOLE
ON-LINE ADAPTER
SINGLE-CHANNEL COMMUNICATION CONTROL

FORMAT



FUNCTION

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 and, when necessary, the method of information transfer according to Table 8-22.

Table 8-22. Description of PDT I/O Control Characters C1 and C2

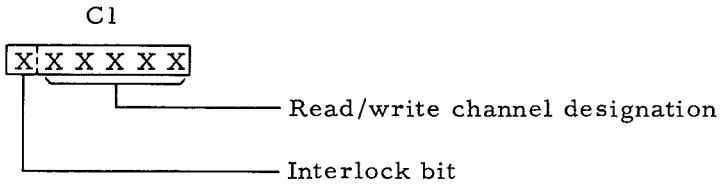
CONTROL CHARACTER	DESCRIPTION
C1	READ/WRITE CHANNEL DESIGNATION: This six-bit character specifies the read/write channel selected to complete the data path.  Interlock Bit: This bit designates whether or not an auxiliary read/write channel will be granted access to main memory by the traffic control. When dealing with RWC1 and RWC1', an interlock bit of zero designates that memory cycle allocation is to be shared by both RWC1 and RWC1'. If the interlock bit is a one, memory access is not granted to RWC1'. (Model 2200 users may apply this same principle to RWC4 and RWC4'.) NOTE: Specific Honeywell manuals and information bulletins define the conditions under which the interlock bit should be used. Read/Write Channel Designation: These five bits specify the read/write channel selected to complete the data path between the main memory and the peripheral control. The octal designation of control character C1 will thus be one of the following configurations, depending upon the RWC selected. Read/Write Channel — Control Character (octal) RWC1 (with interlock) — 51 RWC1 (without interlock) — 11 RWC2 — 12 RWC3 — 13 RWC1' — 15 Model 2200 users may also employ the following configurations. Read/Write Channel — Control Character (octal) RWC4 (with interlock) — 71 RWC4 (without interlock) — 31 RWC5 — 32 RWC6 — 33 RWC4' — 35

Table 8-22 (cont). Description of PDT I/O Control Characters C1 and C2

CONTROL CHARACTER	DESCRIPTION
C2	PERIPHERAL CONTROL DESIGNATION: Control character C2 designates the logical address of the peripheral control to be used in the data transfer. This address depends upon the I/O trunk to which the peripheral control is permanently attached. Model 200: Eight octal addresses may be employed in a Model 200 system containing eight I/O trunks. These addresses are as follows: 00, 01, 02, 03, 40, 41, 42, and 43. In a system equipped with an additional set of eight I/O trunks, the following eight addresses may be used: 04, 05, 06, 07, 44, 45, 46, and 47. Model 1200: Sixteen octal addresses may be employed in a Model 1200 system equipped with 16 I/O trunks. These addresses are as follows: 00, 01, 02, 03, 04, 05, 06, 07, 40, 41, 42, 43, 44, 45, 46, and 47. Model 2200: Sixteen octal addresses may be employed in a Model 2200 system containing 16 I/O trunks. These addresses are as follows: 00, 01, 02, 03, 04, 05, 06, 07, 40, 41, 42, 43, 44, 45, 46, and 47. In a system equipped with an additional set of 16 I/O trunks, the following 16 addresses may be used: 20, 21, 22, 23, 24, 25, 26, 27, 60, 61, 62, 63, 64, 65, 66, and 67. NOTES: 1. Peripheral controls capable of both reading and writing (e.g., a magnetic tape control) must be assigned two addresses — one for reading and one for writing. In this case, the high-order bit of C2 must be 1 for input and 0 for output; the low-order three bits must be the same. 2. For Model 2200 users, the additional set of 16 I/O trunks must be used only in conjunction with RWC4, RWC5, RWC6, and RWC4'.
C3 through Cn	ADDITIONAL PARAMETERS: 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 Table 8-23.

PUNCTUATION MARKS

The execution of this instruction neither affects nor is affected by word marks. However, record marks may terminate the data transfer, depending upon the device used and the operation performed (see the specific Honeywell manuals and information bulletins).

TIMING

Model 200 Processors:

$T = N_i+1$ memory cycles + data transfer time.

Type 1201 Processor:

$T = (N_i-N_c+1)$ memory cycles + $(N_{cn}+3)$ input/output cycles + 1 processor cycle + data transfer time. [1]

Type 2201 Processor:

$T = (N_i-N_c+1)$ memory cycles + N_c alternate memory cycles + data transfer time.

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 Types 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 2201 processor equipped with the Storage Protect Feature (see Appendix E).

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.

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						

[1] The input/output traffic control of the Type 1201 processor gives one out of every four memory cycles unconditionally to the processor for internal operations; this cycle is called the "processor cycle." The three remaining cycles, which can be allocated either to the processor or to read/write channels, are called "input/output cycles."

Table 8-23. 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^1 X	none	none	none	none
	PUNCH	X X	X^2 X	none	none	none	none
See: Honeywell Series 200 Equipment Operators' Manual (DSI-294)							
PAPER TAPE	READ	X X	X^1 X	See Table 8-24 (page 8-114)	none	none	none
	PUNCH	X X	X^2 X	See Table 8-25 (page 8-114)	none	none	none
See: Models 209/210 Paper Tape Equipment (DSI-322)							
PRINTER	PRINT	X X	X^2 X	See Table 8-26 (page 8-114)	none	none	none
See: Honeywell Series 200 Equipment Operators' Manual (DSI-294)							
MAGNETIC TAPE 1/2 INCH	READ FORWARD	X X	X^1 X	6 D (D=tape drive, 0 - 7)[3]	none	none	none
	READ REVERSE (Feature 010 or 011)	X X	X^1 X	2 D (D=tape drive, 0 - 7)[4]	none	none	none
	WRITE	X X	X^2 X	2 D (D=tape drive, 0 - 7)[5]	none	none	none
	SPACE FORWARD	X X	X^1 X	4 D (D=tape drive, 0 - 7)	none	none	none
	BACKSPACE	X X	X^1 X	0 D (D=tape drive, 0 - 7)	none	none	none
	ERASE	X X	X^2 X	0 D (D=tape drive, 0 - 7)	none	none	none
See: Honeywell Series 200 Equipment Operators' Manual (DSI-294)							
MAGNETIC TAPE 3/4 INCH	READ FORWARD	X X	X^1 X	6 D (D=tape drive, 0 - 7)	none	none	none
	READ SUPPRESSING CHANNEL	X X	X^1 X	5 D (D=tape drive, 0 - 7)	C 0 (C=channel to be suppressed)	none	none
	WRITE	X X	X^2 X	6 D (D=tape drive, 0 - 7)	none	none	none
	SKIP WRITE	X X	X^2 X	4 D (D=tape drive, 0 - 7)	none	none	none
	BACKSPACE	X X	X^1 X	0 D (D=tape drive, 0 - 7)	none	none	none
See: Honeywell Series 200 Equipment Operators' Manual (DSI-294)							
RANDOM ACCESS DRUM	SEARCH AND READ	X X	X^1 X	See Table 8-27 (page 8-115)	0 T T T: 9-bit track address numbered 0 - 777 (octal)		S S: Sector address numbered 0 - 47 (octal)
	READ	X X	X^1 X	See Table 8-27 (page 8-115)	none	none	none
	SEARCH AND WRITE	X X	X^2 X	See Table 8-27 (page 8-115)	0 T 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 8-27 (page 8-115)	none	none	none
	READ ADDRESS REGISTER	X X	X^1 X	See Table 8-27 (page 8-115)	none	none	none
See: Model 270 Random Access Drum and Control (DSI-348)							

Table 8-23 (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
MASS MEMORY FILE	LOAD ADDRESS REGISTER	X X	X^2 X	4 X (X=unused)	none	none	none
	STORE ADDRESS REGISTER	X X	X^1 X	4 X (X=unused)	none	none	none
	SEARCH AND READ	X X	X^1 X	2 D (D=device address)	none	none	none
	SEARCH AND WRITE	X X	X^2 X	2 D (D=device address)	none	none	none
	SEARCH AND READ NEXT	X X	X^1 X	3 D (D=device address)	none	none	none
	SEARCH AND WRITE NEXT	X X	X^2 X	3 D (D=device address)	none	none	none
	READ INITIAL	X X	X^1 X	0 D (D=device address)	none	none	none
	READ	X X	X^1 X	1 D (D=device address)	none	none	none
	FORMAT WRITE INITIAL	X X	X^2 X	0 D (D=device address)	none	none	none
	FORMAT WRITE	X X	X^2 X	1 D (D=device address)	none	none	none
See: Mass Memory File Transports and Control (File No. 112.0005.1400.00.01)							
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: Honeywell Series 200 Equipment Operators' Manual (DSI-294)							
ON LINE ADAPTER	TRANSFER ID character to Model 200, Model 1200, or Model 2200 memory.	X X	X X	4 X (X=unused)	none	none	none
	ACCEPT the H-800/1800 instruction defined in the ID register.[6]	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+4 or U+5.[6]	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).[6]	X X	X X	1 U (U = any value from 0 - 7, octal)	none	none	none
	SET the device busy indicator.[6]	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

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.
6. This operation issues initiating and concluding device-ready responses.

Table 8-24. C3 Coding for Type 209 Paper Tape Reader

BIT VALUE	CHANNEL					
	6	5	4	3	2	1
1	Not used	One character per frame	Sense end of record	Check odd parity	Forward	Increment RWC
0	Not used	Two characters per frame	Do not sense end of record	Check even parity	Reverse	Decrement RWC

Table 8-25. C3 Coding for Type 210 Paper Tape Punch

BIT VALUE	CHANNEL					
	6	5	4	3	2	1
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 8-26. C3 Coding for Types 206 and 222 Printers

Type 206		Type 222	
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. Do not print; space to channel xxx.
101111	Do not print; space to the head of the form.	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)

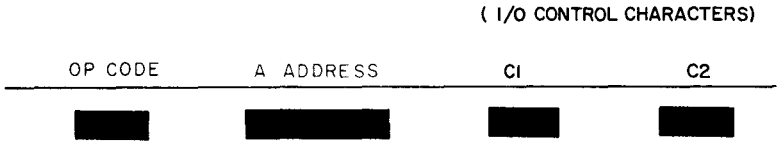
Table 8-27. C3 Coding for Type 270 Random Access Drum

BIT VALUE	CHANNEL					
	6	5	4	3	2	1
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			

PDT PERIPHERAL DATA TRANSFER

For: MULTI-CHANNEL COMMUNICATION CONTROL

FORMAT



FUNCTION

This PDT instruction causes data to be transferred between the Multi-Channel Communication Control and the main memory area designated by the A address. Input/output control character C1 designates the read/write channel through which the data will be transferred, while C2 designates the peripheral control address. Both C1 and C2 are described in Table 8-22 (see page 8-109).

PUNCTUATION MARKS

The execution of this instruction neither affects nor is affected by word marks or record marks.

TIMING

Model 200 Processors:

$T = N_i+1$ memory cycles + data transfer time.

Type 1201 Processor:

$T = (N_i-N_c+1)$ memory cycles + $(N_{cn}+3)$ input/output cycles + 1 processor cycle + data transfer time. [1]

[1] The input/output traffic control of the Type 1201 processor gives one out of every four memory cycles unconditionally to the processor for internal operations; this cycle is called the "processor cycle." The three remaining cycles, which can be allocated either to the processor or to read/write channels, are called "input/output cycles."

Type 2201 Processor:

$T = (N_i - N_c + 1)$ memory cycles + N_c alternate memory cycles + data transfer time.

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 SR1 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 Types 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 2201 processor equipped with the Storage Protect Feature (see Appendix E).

Table 8-28. 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
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 six-line control transmission PDT instructions are listed in Table 8-29, below.	X X	X[2] X	none

NOTES: 1. The high-order bit must be 1.
2. The high-order bit must be 0.

Table 8-29. Type 286 Line Control Instructions

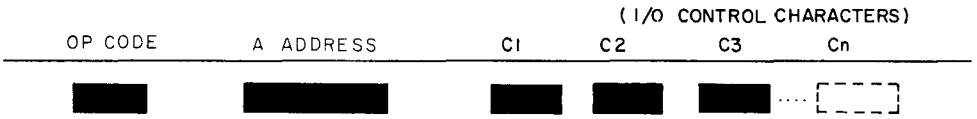
CODE[1] (OCTAL)	INSTRUCTION	DESCRIPTION
10	Transmit last character	Inform the 286 that the last character has been sent from the central processor, and place the control unit in the receive mode for that line (after transmitting last character).
60	Receive clear	Reset the bits of the logic character in the 286 memory. (This instruction should be given when power is first turned on.)
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.

NOTE: The control code is stored in location LOC+1. (The high-order four bits of this location contain the code; the low-order two bits must be 0.)

PCB PERIPHERAL CONTROL AND BRANCH

For: PUNCHED CARD EQUIPMENT
PAPER TAPE EQUIPMENT
PRINTER
MAGNETIC TAPE EQUIPMENT
RANDOM ACCESS DRUM
MASS MEMORY FILE
CONSOLE
ON-LINE ADAPTER
SINGLE-CHANNEL COMMUNICATION CONTROL

FORMAT



FUNCTION

The Peripheral Control and Branch instruction can initiate two types of operations: (1) test operations, and (2) control operations.

1. Test operations test the status of the peripheral control to which the PCB instruction is issued (e.g., test for a "busy" status, test to determine if an error is present, etc.)
2. Control operations set the peripheral control to perform a specific control function(e.g., reject error cards when addressed to card controls, rewind the tape when addressed to a magnetic tape control, etc.)

The A address of a PCB instruction specifies a main memory location to which the machine branches if the test conditions specified by C3 through Cn are present. If the PCB instruction is initiating a control operation, the A address specifies the main memory location to which a branch is made if the peripheral device is unavailable.

C1 designates the read/write channel. The function of this character is the same as its function for a PDT instruction; see Table 8-22 (page 8-109).

C2 designates the peripheral control. The function of this character is the same as its function for a PDT instruction; see Table 8-22 (page 8-109).

Control characters C3 through Cn designate the control and test operations. 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 Table 8-30 (see page 8-119).

PUNCTUATION MARKS

The execution of this instruction neither affects nor is affected by word marks or record marks.

TIMING

Model 200 Processors:

T = N_i+1 memory cycles if no branch condition exists.

T = N_i+2 memory cycles if a branch occurs.

Type 1201 Processor:

T = (N_i-N_c+1) memory cycles + N_c input/output cycles.[1]

Type 2201 Processor:

T = (N_i-N_c+1) memory cycles + N_c alternate memory cycles.

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR	
NXT	A	B_p	NO BRANCH
JI (A)	A	NXT	BRANCH

[1] The input/output traffic control of the Type 1201 processor gives one out of every four memory cycles unconditionally to the processor for internal operations; this cycle is called the "processor cycle." The three remaining memory cycles, which can be given either to the processor or to read/write channels, are called "input/output cycles.

NOTES

1. Control character C1 tests the status of a read/write channel. If an RWC status test is not desired, C1 must contain zeros.
2. The PCB op code is a "privileged" op code when used in a Type 2201 processor equipped with the Storage Protect Feature (see Appendix E).

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
				PCB	STOP, ØØ, 41, 27, 21

Table 8-30. Summary of PCB I/O Control Characters

TEST AND CONTROL OPERATIONS			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^2 X	1 0
	Branch to A address if punch-check error		X X	X^2 X	4 1
	Branch to A address if device unavailable. If available, set control unit to:	Punch Hollerith code[4]	X X	X^2 X	2 7
		Punch special code	X X	X^2 X	2 6
		Punch direct transcription code (feature 064)	X X	X^2 X	2 5
		Generate busy signal if punch-check error	X X	X^2 X	2 3
		Reject cards with punch-check error	X X	X^2 X	2 1
	Turn the control allow function OFF		X X	X^2 X	7 0
	Turn the control allow function ON		X X	X^2 X	7 1
	Turn the control interrupt function OFF		X X	X^2 X	7 4
	Branch to A address if the control interrupt function is ON		X X	X^2 X	7 5

Table 8-30 (cont). Summary of PCB I/O Control Characters

	TEST AND CONTROL OPERATIONS		PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
TYPE 214-2 CARD READER/CARD PUNCH	Branch to A address if device busy		X X	X[3] X	1 0
	Branch to A address if 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 error cards[4]	X X	X[3] X	2 7
		Punch special code	X X	X[3] X	2 6
		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 punch-check error	X X	X[3] X	2 3
		Reject cards with illegal punches	X X	X[3] X	2 2
		Reject cards with punch-check error	X X	X[3] X	2 1
		Operate in punch-feed read mode	X X	X[3] X	2 0
	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
TYPE 223 CARD READER	Branch to A address if device busy		X X	X[1] X	1 0
	Branch to A address if cycle check error		X X	X[1] X	4 1
	Branch to A address if illegal punch		X X	X[1] 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[1] X	2 7
		Read special code	X X	X[1] X	2 6
		Read direct transcription code (feature 044)	X	X[1] X	2 5

Table 8-30 (cont). Summary of PCB I/O Control Characters

TEST AND CONTROL OPERATIONS			PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
TYPE 223 CARD READER (cont)	Branch to A address if device unavailable. If available, set control unit to:	Reject cards with cycle check error	X X	X[1] X	2 1
		Reject cards with illegal punches	X X	X[1] X	2 2
		Generate busy signal if cycle check error	X X	X[1] X	2 3
		Generate busy signal if illegal punch	X X	X[1] X	2 4
	Turn the control allow function OFF		X X	X[1] X	7 0
	Turn the control allow function ON		X X	X[1] X	7 1
	Turn the control interrupt function OFF		X X	X[1] X	7 4
	Branch to A address if the control interrupt function is ON		X X	X[1] X	7 5
See: Honeywell Series 200 Equipment Operators' Manual (DSI-294)					
TYPE 224-1 CARD PUNCH	Branch to A address if device busy		X X	X[2] X	1 0
	Branch to A address if echo-check error		X X	X[2] X	4 1
	Branch to A address if device unavailable. If available, set control unit to:	Punch Hollerith code[4]	X X	X[2] X	2 7
		Punch special code	X X	X[2] X	2 6
		Punch direct transcription code[5] (feature 064)	X X	X[2] X	2 5
		Generate busy signal if echo-check error	X X	X[2] X	2 3
		Reject cards with echo-check errors	X X	X[2] X	2 1
	Turn the control allow function OFF		X X	X[2] X	7 0
	Turn the control allow function ON		X X	X[2] X	7 1
	Turn the control interrupt function OFF		X X	X[2] X	7 4
	Branch to A address if the control interrupt function is ON		X X	X[2] X	7 5

Table 8-30 (cont). Summary of PCB I/O Control Characters

	TEST AND CONTROL OPERATIONS		PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
TYPE 224-2 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[5] (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	X X	X[3] X	2 2
		Reject cards with echo-check or read registration errors	X X	X[3] X	2 1
		Operate in punch-feed read mode	X X	X[3] X	2 0
	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: Honeywell Series 200 Equipment Operators' Manual (DSI-294)					
TYPE 227 CARD READER	Branch to A address if device busy		X X	X[1] X	1 0
	Branch to A address if hole-count error		X X	X[1] X	4 1
	Branch to A address if illegal punch		X X	X[1] 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[1] X	2 7

Table 8-30 (cont). Summary of PCB I/O Control Characters

TEST AND CONTROL OPERATIONS			PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
TYPE 227 CARD READER (cont)	Branch to A address if device unavailable. If available, set control unit to:	Read special code	X X	X[1] X	2 6
		Read direct transcription code (feature 040)	X X	X[1] X	2 5
		Reject cards with hole-count errors	X X	X[1] X	2 1
		Reject cards with illegal punches	X X	X[1] X	2 2
		Generate busy signal if hole-count error	X X	X[1] X	2 3
		Generate busy signal if illegal punch	X X	X[1] X	2 4
		Place previously read card in middle stacker (feature 017)	X X	X[1] X	3 1
		Place previously read card in the read eject stacker (feature 017-1)	X X	X[1] X	3 2
	Turn the control allow function OFF		X X	X[1] X	7 0
	Turn the control allow function ON		X X	X[1] X	7 1
	Turn the control interrupt function OFF		X X	X[1] X	7 4
	Branch to A address if the control interrupt function is ON		X X	X[1] X	7 5
See: Honeywell Series 200 Equipment Operators' Manual (DSI-294)					
TYPE 227 CARD PUNCH	Branch to A address if device busy		X X	X[2] X	1 0
	Branch to A address if hole-count error[6] (feature 061)		X X	X[2] 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[2] X	2 7
		Punch special code	X X	X[2] X	2 6
		Punch direct transcription code[5] (feature 060)	X X	X[2] X	2 5

Table 8-30 (cont). Summary of PCB I/O Control Characters

	TEST AND CONTROL OPERATIONS		PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
TYPE 227 CARD PUNCH (cont)	Branch to A address if device unavailable. If available, set control unit to:	Reject cards with illegal punches (feature 052)	X X	X^2 X	2 2
		Reject cards with hole-count errors[5] (feature 061)	X X	X^2 X	2 1
		Punch-feed read operations[5] (feature 062)	X X	X^2 X	2 0
		Place previously punched card in middle stacker[5] (feature 017)	X X	X^2 X	3 1
		Place previously punched card in the punch eject stacker (feature 017-1)	X X	X^2 X	3 2
	Turn the control allow function OFF		X X	X^2 X	7 0
	Turn the control allow function ON		X X	X^2 X	7 1
	Turn the control interrupt function OFF		X X	X^2 X	7 4
	Branch to A address if the control interrupt function is ON		X X	X^2 X	7 5
See: Honeywell Series 200 Equipment Operators' Manual (DSI-294)					
TYPE 209 PAPER TAPE READER	Branch to A address if device busy		X X	X^1 X	1 X (X=unused)
	Branch to A address if parity error		X X	X^1 X	4 X (X=unused)
	Branch to A address if device unavailable. If available, set control unit to:	Rewind the tape (reverse direction)	X X	X^1 X	3 0
		Runout the tape (forward direction)	X X	X^1 X	3 2
	Turn the control allow function OFF		X X	X^1 X	7 0
	Turn the control allow function ON		X X	X^1 X	7 1
	Turn the control interrupt function OFF		X X	X^1 X	7 4
	Branch to A address if the control interrupt function is ON		X X	X^1 X	7 5
See: Models 209/210 Paper Tape Equipment (DSI-322)					

Table 8-30 (cont). Summary of PCB I/O Control Characters

	TEST AND CONTROL OPERATIONS	PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
TYPE 210 PAPER TAPE PUNCH	Branch to A address if device busy	X X	X^2 X	1 0
	Turn the control allow function OFF	X X	X^2 X	7 0
	Turn the control allow function ON	X X	X^2 X	7 1
	Turn the control interrupt function OFF	X X	X^2 X	7 4
	Branch to A address if the control interrupt function is ON	X X	X^2 X	7 5
See: Models 209/210 Paper Tape Equipment (DSI-322)				
TYPE 206 PRINTER	Branch to A address if device busy	X X	X^2 X	1 0
	Branch to A address if print error	X X	X^2 X	4 0
See: Honeywell Series 200 Equipment Operators' Manual (DSI-294)				
TYPE 222 PRINTER	Branch to A address if device busy	X X	X^2 X	1 0
	Branch to A address if print error	X X	X^2 X	4 0
	Branch to A address if formatting is complete	X X	X^2 X	2 0
	Branch to A address if end of form	X X	X^2 X	0 1
	Branch to A address if channel eight	X X	X^2 X	0 2
	Turn the control allow function OFF	X X	X^2 X	7 0
	Turn the control allow function ON	X X	X^2 X	7 1
	Turn the control interrupt function OFF	X X	X^2 X	7 4
	Branch to A address if the control interrupt function is ON	X X	X^2 X	7 5
See: Honeywell Series 200 Equipment Operators' Manual (DSI-294)				

Table 8-30 (cont). Summary of PCB I/O Control Characters

	TEST AND CONTROL OPERATIONS	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)
	Branch to A address if 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: Honeywell Series 200 Equipment Operators' Manual (DSI-294)				
MAGNETIC TAPE UNITS 3/4 INCH	Rewind	X X	X^2 X	2 D (D=tape drive, 0 - 7)
	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)

Table 8-30 (cont). Summary of PCB I/O Control Characters

TEST AND CONTROL OPERATIONS		PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
MAGNETIC TAPE UNITS 3/4 INCH (cont)	Branch to A address if beginning of tape	X X	X[1] X	6 D (D=tape drive, 0 - 7)
	Branch to A address if end of tape	X X	X[2] X	6 D (D=tape drive, 0 - 7)
	Branch to A address if "long check" error is detected	X X	X[2] X	5 X (X=unused)
	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 203A/204A Three-Quarter Inch Magnetic Tape Systems (DSI-342)				
TYPE 270 RANDOM ACCESS DRUM	Branch to A address if device busy[5]	X[1] X	X X	0 X or 1 X (X=unused)
	Branch to A address if error indicator is on	X[1] X	X X	4 X (X=unused)
	Turn the control allow function OFF	X[1] X	X X	7 0
	Turn the control allow function ON	X[1] X	X X	7 1
	Turn the control interrupt function OFF	X[1] X	X X	7 4
	Branch to A address if the control interrupt function is ON	X[1] X	X X	7 5
See: Model 270 Random Access Drum and Control (DSI-348)				
MASS MEMORY FILE	Branch to A address if control busy	X[1] X	X[2] X	1 X (X=unused)
	Branch to A address if device busy	X[1] X	X[2] X	0 D (D=device address)
	Branch to A address if general exception	X[1] X	X[2] X	5 X (X=unused)
	Branch to A address if device unavailable	X[1] X	X[2] X	4 D (D=device address)
	Branch to A address if TLL flag	X[1] X	X[1] X	5 X (X=unused)

Table 8-30 (cont). Summary of PCB I/O Control Characters

TEST AND CONTROL OPERATIONS			PCB I/O CONTROL CHARACTERS		
			C1	C2	C3 through Cn
MASS MEMORY FILE (cont)	Branch to A address if specified transport is busy. If not busy, set control unit to:	Seek out the strip (specified by C5 and C6) in the cartridge (specified by C4)	X[1] X	X[2] X	C3: 2 D (D=device address) C4: 0 0 for Types 251 and 252; 0 0 to 0 4 for Type 253 C5 and C6: 0000 to 0777
		Return the tape strip currently on the read/write drum (if any) to the cartridge.	X[1] X	X[2] X	3 D (D=device address)
		Return the tape strip currently on the read/write drum to the cartridge. Simultaneously, if another strip is on the waiting platform (or is in the process of being placed on the platform), that strip is also returned to the cartridge.	X[1] X	X[1] X	3 D (D=device address)
	Turn the control allow function OFF		X[1] X	X[6] X	7 0
	Turn the control allow function ON		X[1] X	X[6] X	7 1
	Turn the transport allow function OFF		X[1] X	X[6] X	7 2
	Turn the transport allow function ON		X[1] X	X[6] X	7 3
	Turn the control interrupt function OFF[7]		X[1] X	X[6] X	7 4
	Branch to A address if the control interrupt function is ON		X[1] X	X[6] X	7 5
	Turn the transport interrupt function OFF		X[1] X	X[6] X	7 6
	Branch to A address if the transport interrupt function is ON		X[1] X	X[6] X	7 7
See: Mass Memory File Transports and Control (File No. 112.0005.1400.00.01)					
TYPE 220-1 CONSOLE	Branch to A address if device busy		X X	X[2] X	1 0
See: Honeywell Series 200 Equipment Operators' Manual (DSI-294)					

Table 8-30 (cont). Summary of PCB I/O Control Characters

	TEST AND CONTROL OPERATIONS	PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
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: Honeywell Series 200 Equipment Operators' Manual (DSI-294)				
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
TYPE 212 ON-LINE ADAPTER	Branch to A address if device busy	X[1] X	X[3] X	0 X or 1 X (X=unused)
	Branch to A address if data transfer is in progress	X[1] X	X[3] X	7 X (X=unused)
	Branch to A address if error or incomplete indicator is set	X[1] X	X[3] X	4 X
	Branch to A address if parity error is stored	X[1] X	X[3] X	5 X (X=unused)
	Branch to A address if incomplete error is stored	X[1] X	X[3] X	6 X (X=unused)
	Place control character C4 in the ID register if data transfer is not in progress	X[1] 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[1] X	X[3] X	3 X (X=unused)

Table 8-30 (cont). Summary of PCB I/O Control Characters

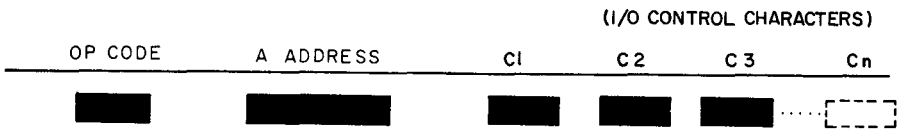
	TEST AND CONTROL OPERATIONS	PCB I/O CONTROL CHARACTERS		
		C1	C2	C3 through Cn
TYPE 281 SCCC	Branch to A address if device busy	X X	X[3] X	1 0
	Branch to A address if parity error	X X	X[3] X	4 0
	Branch to A address if error other than parity error	X X	X[3] X	5 0
	Branch to A address if the 281 is in 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[3] X	7 0
	Turn the allow function ON	X X	X[3] X	7 1
	Turn the interrupt function OFF	X X	X[3] X	7 4
	Branch to A address if allow and interrupt functions are ON	X X	X[3] X	7 5

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 transport are automatically turned on when a "not busy" status is reached by the control or transport, respectively.
8. For program interruption in the 201 or 201-1 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 or 201-1 central processor; C3 control characters 76 and 77 perform no operations with other central processors. In those cases where the 201 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.

PCB | PERIPHERAL CONTROL AND BRANCH | For: MULTI-CHANNEL COMMUNICATION CONTROL

FORMAT



FUNCTION

The Peripheral Control and Branch instruction can initiate two types of operations: (1) test operations, and (2) control operations.

1. Test operations test the status of the peripheral control to which the PCB instruction is issued.
2. Control operations set the peripheral control to perform a specific control function.

The A address of a PCB instruction specifies a main memory location to which the machine branches if the test conditions specified by C3 through Cn are present. If the PCB instruction is initiating a control operation, the A address is immaterial (although it must still be present in the instruction).

C1 designates the read/write channel. The function of this character is the same as its function for a PDT instruction; see Table 8-22 (page 8-109).

C2 designates the peripheral control. The function of this character is the same as its function for a PDT instruction; see Table 8-22 (page 8-109).

Control characters C3 through Cn designate the control and test operations. A summary of coding for these characters may be found in Table 8-31 (see page 8-132).

PUNCTUATION MARKS

The execution of this instruction neither affects nor is affected by word marks or record marks.

TIMING

Model 200 Processors:

$T = N_i+1$ memory cycles if no branch condition exists.

$T = N_i+2$ memory cycles if a branch occurs.

Type 1201 Processor:

$T = (N_i-N_c+1)$ memory cycles + N_c input/output cycles.[1]

Type 2201 Processor:

$T = (N_i-N_c+1)$ memory cycles + N_c alternate memory cycles.

[1] The three out of every four memory cycles which can be allocated to either the processor or to read/write channels are referred to as "input/output cycles."

ADDRESS REGISTERS AFTER OPERATION

SR	AAR	BAR	
NXT	A	B_p	NO BRANCH
JI (A)	A	NXT	BRANCH

NOTES

1. Control character C1 tests the status of a read/write channel. If an RWC status test is not desired, C1 must contain zeros.
2. The PCB op code is a "privileged" op code when used in a Type 2201 processor equipped with the Storage Protect Feature (see Appendix E).

Table 8-31. Summary of PCB I/O Control Characters for Type 286 Multi-Channel Communication Control

TEST AND CONTROL OPERATIONS	PCB I/O CONTROL CHARACTERS		
	C1	C2	C3
Branch to A address if device busy. If not busy, set the 286 to stop scanning and continue the program in sequence[1]	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
Branch to A address if the interrupt was due to the 286 requesting service	X X	X X	7 5
NOTE: 1. The busy test has no significance for the Type 286.			

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	1710	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

OCTAL ADDRESS (Control Panel or Console Addressing)		VARIANT CHARACTER (LCR & SCR Instructions)	REGISTER	
200	1200 & 2200		Mnemonic	Function
01	01	01	CLC1	Read/Write Channel 1 — Current Location Counter
02	02	02	CLC2	Read/Write Channel 2 — Current Location Counter
03	03	03	CLC3	Read/Write Channel 3 — Current Location Counter
05	05	05	CLC1'	Read/Write Channel 1' — Current Location Counter
11	11	11	SLC1	Read/Write Channel 1 — Starting Location Counter
12	12	12	SLC2	Read/Write Channel 2 — Starting Location Counter
13	13	13	SLC3	Read/Write Channel 3 — Starting Location Counter
15	15	15	SLC1'	Read/Write Channel 1' — Starting Location Counter
	21[1]	21	CLC4	Read/Write Channel 4 — Current Location Counter
	22[1]	22	CLC5	Read/Write Channel 5 — Current Location Counter
	23[1]	23	CLC6	Read/Write Channel 6 — Current Location Counter
	25[1]	25	CLC4'	Read/Write Channel 4' — Current Location Counter
	31[1]	31	SLC4	Read/Write Channel 4 — Starting Location Counter
	32[1]	32	SLC5	Read/Write Channel 5 — Starting Location Counter
	33[1]	33	SLC6	Read/Write Channel 6 — Starting Location Counter
	35[1]	35	SLC4'	Read/Write Channel 4' — Starting Location Counter

Table B-1 (cont). Control Register Designations

OCTAL ADDRESS (Control Panel or Console Addressing)		VARIANT CHARACTER (LCR & SCR Instructions)	REGISTER	
200	1200 & 1200		Mnemonic	Function
04	64	64	CSR	Change Sequence Register
06	66	66	EIR	External Interrupt Register
14	74	67	AAR	A-Address Register
10	70	70	BAR	B-Address Register
n/a	76	76	IIR	Internal Interrupt Register
17	77	77	SR	Sequence Register
n/a	41-43	n/a	AC0	Floating-Point Accumulator 0
n/a	45-47	n/a	AC1	Floating-Point Accumulator 1
n/a	51-53	n/a	AC2	Floating-Point Accumulator 2
n/a	55-57	n/a	AC3	Floating-Point Accumulator 3

[1] 2200 only.

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
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.

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 Record 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
51*	Word mark, A bit is 1
52*	Word mark, B bit is 1
53*	Word mark, B and A bits are 11
54*	Word mark, B and A bits are 00
55*	Word mark, Positive sign
56*	Word mark, Negative sign
57*	Word mark, B and A bits are 11

Table B-5 (cont). Branch on Character Condition (BCC) Conditions

VARIANT CHARACTER (Octal)	BRANCH CONDITION
60*	Item mark
61*	Item mark, A bit is 1
62*	Item mark, B bit is 1
63*	Item mark, B and A bits are 11
64*	Item mark, B and A bits are 00
65*	Item mark, Positive sign
66*	Item mark, Negative sign
67*	Item mark, B and A bits are 11
70*	Unconditional
71*	Word mark or A bit is 1
72*	Word mark or B bit is 1
73*	Word mark or B and A bits are 11
74*	Word mark or B and A bits are 00
75*	Word mark or Positive sign
76*	Word mark or Negative sign
77*	Word mark or B and A bits are 11

*Valid only on systems equipped with the Advanced Programming Feature (Feature 010 or 011).

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	! [2]
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 0	R or R, 0[1]	011111	37	?[2]		0, 8, 7	111111	77	¢ [2]

[1] Special Code. This card code-central processor code equivalency is effective 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).

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

APPENDIX C

INSTRUCTION SUMMARY

Table C-1. Instruction Summary of Appendix C

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								
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.	8-16
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.	8-18
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.	8-20
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.	8-21
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.	8-23
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.	8-24
M	26	R, 6	F	Decimal Multiply	See page 8-28.	a. M/A, B b. M/A c. M/	Preserves B.	A and B fields.	Both word marks.	Yes.	8-26
D	27	R, 7	G	Decimal Divide	See page 8-31...	a. D/A, B b. D/A c. D/	Preserves B.	A operand (divisor).	A-operand word mark.	Yes.	8-29
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.	8-34
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.	8-35
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.	8-37
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.	8-38
B	65	0, 5	V	Branch (Unconditional)	N_i+2	a. B/A	Bypasses B.	None.	n/a	No.	8-40
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]	8-41
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 d. BCC/	Preserves B.	None.	Single-character operation.	Yes.	8-45
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]	8-49
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.	8-51
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.	8-54
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.	8-55
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.	8-56
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.	8-58
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.	8-59
NOP	40		-	No Operation	3[12]	a. NOP/	Bypasses A and B.	None.	n/a	No.	8-61
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.	8-62
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.	8-63
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]	8-65

Table C-1 (cont). Instruction Summary of Appendix C

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)											
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]	8-67
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]	8-69
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]	8-72
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 8-74.	See page 8-74.	Yes. [8]	8-74
MAT	60	8, 5		Move and Translate	N_i+3N_a [13]	a. MAT/A,B,V_1,V_2	See page 8-77.	A operand.	Word mark in A operand or in table.	No.	8-77
MIT	62	0, 2	S	Move Item and Translate	$N_i+N_a+2(N_{ut})(NB_u)$ [14]	a. MIT/A,B, V_1, V_2, V_3	See page 8-80.	None.	A-operand item mark or word mark in table.	No.	8-80
LIB	77	0, 8, 7		Load Index/ Barricade Register	N_i+3	a. LIB/A b. LIB/	Preserves B.	None.	Single-character operation.	Yes.	8-84
SIB	76	0, 8, 6		Store Index/ Barricade Register	N_i+3	a. SIB/A b. SIB/	Preserves B.	None.	Single-character operation.	Yes.	8-87
INTERRUPT CONTROL INSTRUCTIONS											
SVI	46	X, 6	O	Store Variant and Indicators	$N_i+1+N_a+N_j$ [15]	a. SVI/V	Bypasses A and B.	None.	Word mark of next instruction.	No.	8-90
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.	8-93
MC	44	X, 4	M	Monitor Call	N_i+2 [4]	a. MC/	Bypasses A and B.	None.	Word mark of next instruction.	No.	8-95
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.	8-97
EDITING INSTRUCTION											
MCE	74	0, 8, 4	%	Move Characters and Edit	$N_i+1+N_a+2N_b+2X+2Y$	a. MCE/A, B [7] b. MCE/A c. MCE/	Preserves B.	A operand and B operand (see page 8-104).	See page 8-104.	No.	8-102
INPUT/OUTPUT INSTRUCTIONS											
PDT	66	0, 6	W	Peripheral Data Transfer	See page 8-111.	a. PDT/A, C_1,.. C_n	Bypasses B.	None.	Record mark in memory or unit record length.	No.	8-108 and 8-115
PCB	64	0, 4	U	Peripheral Control and Branch	See page 8-118.	a. PCB/A, C_1,.. C_n	Bypasses B.	None.	n/a	No.	8-117 and 8-131.

[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 being executed in a Type 1201 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 being 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 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 processor.

[13] Add four memory cycles to this formula if the instruction is executed in a Type 2201 processor.

[14] Add two memory cycles to this formula if the instruction is executed in a Type 1201 processor. Add four cycles to the formula if the instruction is executed in a Type 2201 processor.

[15] Add one memory cycle to this formula if the instruction is executed in a Type 1201 processor. Add two cycles to the formula if the instruction is executed in a Type 2201 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 processor.

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 or 2201 with the Storage Protect Feature (see Appendix E), by an internal interrupt source.

EXTERNAL INTERRUPT

An external interrupt signal can be generated by any one of three sources:

1. a peripheral control (including data communication controls);
2. the operator's control panel or console; or
3. the Monitor Call instruction (see page 8-95).

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 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, or 2201 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.

The interrupt signal is maintained by the source until the processor responds by taking the following actions:

1. The current status of the arithmetic, comparison, address mode, and trap mode indicators are stored in the auxiliary indicators register (AIR).

2. The arithmetic indicators are cleared.
3. The processor enters the three-character, non-trap mode.
4. The contents of SR and EIR are interchanged, and the program branches to the instruction whose op code address was previously stored in EIR.
5. The processor enters the external interrupt mode.

INTERNAL INTERRUPT

An internal interrupt signal is generated only by a Type 1201 or 2201 processor equipped with the Storage Protect Feature and is caused by a "violation" of storage protection. (The nature of storage protect violations -- II 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 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 or 2201 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]

The processor responds to an internal interrupt signal as follows:

1. The processor enters the three-character, non-trap mode.
2. The contents of SR and IIR are interchanged, and the program branches to the instruction whose op code address was previously stored in IIR.
3. The processor enters the internal interrupt mode.

[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 processor responds to these sources is determined by the program (i.e., according to the programmer-established sequence of interrupt source tests).

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 8-90 through 8-97) 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, etc.); and (b) information required to identify the interrupt source.
2. The RestoreVariant 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 the interrupted program.

The fourth interrupt control instruction — Monitor Call (MC) — causes an external interruption and is therefore 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.

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				AAR	CEQU	#1C67	A-ADDRESS REGISTER
2				BAR	CEQU	#1C7Ø	B-ADDRESS REGISTER
3				MAX	CEQU	#1CØ4	MAXIMUM ADD. MODE FOR C.P. IS 4
4				ALLS	CEQU	#1C77	ALL INDICATORS
5				ALLR	CEQU	#1C37	ALL BUT INTERRUPT INDICATORS
6					ADMODE	MAX	SET MAXIMUM ADDRESSING MODE
7				RESTOR	RVI	ENTER+2, ALLR	RESTORE ALL BUT INT. IND.
8				EXIT	RNM	Ø,Ø	EXIT WITH AAR + BAR RESTORED
9				ENTER	SVI	ALLS	ENTER AND STORE ALL INDICATORS
10					RESV	6	RESERVE STORAGE FOR INDICATORS
11					CAM	MAX	ENTER MAXIMUM ADDRESS MODE
12					SCR	EXIT+4, AREG	SAVE AAR
13					SCR	EXIT+8, BREG	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 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 tagged START was previously stored in IIR and that the maximum addressing mode of the processor is the four-character mode.

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				AAR	CEQU	#1C67	A-ADDRESS REGISTER
2				BAR	CEQU	#1C7Ø	B-ADDRESS REGISTER
3				MAX	CEQU	#1CØ4	MAXIMUM ADD. MODE FOR CP IS 4
4				INDS	CEQU	#1C73	ALL BUT AIR INDICATORS
5				INDR	CEQU	#1C33	ALL BUT AIR AND INT. INDICATORS
6				SAVEA	DCW	#4C	TEMPORARY STORAGE FOR AAR
7				SAVEB	DCW	#4C	TEMPORARY STORAGE FOR BAR
8					ADMODE	MAX	SET MAXIMUM ADDRESSING MODE
9				RESTOR	LCR	SAVEA,AAR	RESTORE AAR
10					LCR	SAVEB,BAR	RESTORE BAR
11					RVI	START+2,INDR	RESTORE ALL BUT AIR AND INT. IND.
12					RNM		EXIT
13				START	SVI	INDS	ENTER AND STORE ALL BUT AIR IND.
14					RESV	5	STORAGE FOR ALL BUT AIR IND.
15					CAM	MAX	ENTER MAXIMUM ADDRESSING MODE
16					SCR	SAVEA,AAR	SAVE AAR
17					SCR	SAVEB,BAR	SAVE BAR
18							
19							
20							INTERNAL
21							INTERRUPT ROUTINE
22							
23							
24					B	RESTOR	BRANCH TO RESTOR AND EXIT

Figure D-2. Sample Coding For Internal Interrupt Routine

The initial and concluding instructions in an internal interrupt routine are similar to those in an external interrupt routine, except that the SVI instruction 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.

Another difference between the coding of the two routines is that the RNM instruction is coded in the internal interrupt routine in the format RNM/ (i.e., no address portions). The stored contents of AAR and BAR are therefore restored by two LCR instructions, issued in the maximum addressing mode of the processor, which immediately precede the RVI instruction.

APPENDIX E

STORAGE PROTECT FEATURE

When the Type 1201 or 2201 processor is equipped with the Storage Protect Feature, the main memory can be logically divided into two distinct areas: an "open" area and a "protected" area. When storage protection is in effect, the contents of the protected area are shielded from unintentional interference by a program stored in the open memory area. The boundaries of the protected memory area are specified as follows:

1. The lower boundary is set by the Load Index/Barricade Register instruction (LIB) which specifies the number of a 4,096-character memory bank. This number is loaded into the index/barricade register and is the number of the bank whose leftmost location is the boundary of the protected area.
2. The upper boundary of the protected area is the rightmost location of the main memory.

Index registers 16 through 30 are contained in the leftmost 60 locations of the 4,096-character bank specified in the LIB instruction. Thus, the locations of these registers are redefined when the contents of the index/protect register are altered by an LIB instruction.

The following conditions must be present for storage protection to be in effect:

1. The protect indicator is on. (The protect indicator can be set by the execution of the Restore Variant and Indicators instruction -- see page 8-93.)
2. The processor is neither in the external interrupt mode (see Appendix D) nor in the internal interrupt mode (see below).

INTERNAL INTERRUPT

When storage protection is in effect (i.e., the two conditions specified above are present), certain operations are defined as "violations" of that protection. Such violations are discussed further in this appendix. The violation causes an 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., the processor is executing, without manual intervention, stored-program instructions under control of SR).
2. The processor is about to extract an op code.
3. A memory cycle is allocated to the processor.
4. The processor is neither in the external interrupt mode nor in the internal interrupt mode, and no peripheral or control panel interrupt signal is being received.

The activation of the internal interrupt mode is similar, but not identical, to the processor's actions taken when an external interrupt signal is received (see Appendix D). Three basic differences exist between the two interrupt modes. First, a unique control memory location -- the internal interrupt register (IIR) -- is used to contain the address of the subroutine which services the internal interrupt. (IIR is designated by a variant character of 46_8 in a Store Control Registers or Load Control Registers instruction — see page 8-65.) Secondly, the processor is still subject to being interrupted by an external interrupt while it is in the internal interrupt mode.

The third difference between the internal and external interrupt modes is that no processor indicators are stored or changed when the internal interrupt mode is entered; the handling of processor indicators in the internal interrupt mode is the programmer's responsibility (using the Store Variant and Indicators and the Restore Variant and Indicators instructions — see pages 8-90).

VIOLATIONS OF STORAGE PROTECTION

The following operations are violations of storage protection:

1. An attempt to transfer information internally to a main memory location contained in the protected memory area (i.e., not a peripheral transfer attempt).

   Although information transfers to the protected memory area cause a violation, there is no restriction on the transfer of information from the protected area. In particular, the protected index registers (index registers 16 through 30) can be used for indexing by programs in either the open or protected areas. However, modification of the contents of these registers is inhibited and causes a violation.

   When either of the above-mentioned violations occurs, the II address violation indicator is set. The instruction which causes the violation proceeds normally in all other respects, and the internal interrupt occurs only after the completion of such an instruction.

2. An attempt to extract a Peripheral Data Transfer (PDT) instruction whose effective A address references a protected memory location.[1] Once it is determined that the effective A address references a protected main memory location, no operation is performed (i.e., the specified read/write channel is not tested, and the specified peripheral control is not addressed). The II address violation indicator is set, the contents of SR are advanced to the next sequential op code, and the instruction is terminated without ever having been executed. The internal interrupt occurs at the completion of the instruction.

[1] A PDT instruction is one of eight instructions whose execution is normally prohibited when storage protection is in effect. However, the proceed indicator can be set to permit the execution of this instruction (see page E-3). Thus, for a PDT instruction to be extracted (and therefore to reach the stage where the A address of the instruction is tested), the proceed indicator must first be set to allow the extraction of the instruction.

Note that a PDT instruction is checked for a possible violation during the extraction of that instruction, while a non-peripheral instruction is checked during its execution (as in 1., above). If a PDT instruction passes this test during its extraction, it is free to be executed and thereby cause data to be transferred. If the record being transferred extends into the protected memory area, a violation does not occur. To prevent such a record (i.e., one whose effective A address does not reference the protected area, but a portion of the record transferred extends into the protected area), record marks should be set in both the first and last locations of the protected memory area.

3. An attempt to transfer information from a main memory location that is within the addressing capacity of the memory address register of the user's processor, but which is greater than the capacity of the main memory actually present in the machine.[1] Such an addressing attempt would normally cause the machine to halt due to a parity-check error. However, when storage protect is in effect, such an error does not cause a halt, nor is data transferred into the memory with "bad parity." Instead, the II address violation indicator is set, and the internal interrupt occurs at the completion of the instruction.

4. An attempt to execute an instruction whose op code is: (a) not defined for the Series 200; (b) not recognized by the user's processor; or (c) prohibited when storage protection is in effect. Prohibited op codes under this condition are the following:

   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 detection of any of the above-listed op codes sets the op code violation indicator and causes the contents of SR to be set back to the location of the op code which was the offender. The operation is terminated, and the internal interrupt occurs subject to the other conditions whose presence is required for the internal interrupt to occur (see page E-1).

PROCEED INDICATOR

The proceed indicator can be turned on through the execution of the Restore Variant and Indicators (RVI) instruction (see page 8-93). When the proceed indicator is on, one instruction is permitted to be executed in the non-interrupt mode without : (1) op code checking; or (2) item mark trapping. In other words, both op code checking and item mark

[1] The memory address register contains as many bits as are necessary to address the memory of an individual processor. Thus, a processor whose main memory capacity is 32,768 characters contains 15 bits in the memory address register; a memory capacity of 65,536 characters requires 16 bits in the register. If a processor has a memory capacity of, for example, 49,152 characters, the memory address register must still contain 16 active bits in order to address memory locations above 32,768. In this case, the capacity of the main memory actually present in the processor (49,152) is less than the addressing capacity of the memory address register (65,536). Thus, a location can be addressed which is within the range of the register but which is not actually present in the user's system.

trapping are overruled by a proceed indicator which is turned on. The proceed indicator is turned off by the execution of any instruction in the non-interrupt mode.

The proceed indicator may also be used to enforce the A address checking of a PDT instruction issued in the interrupt mode (either the external or internal interrupt mode). Thus, when the following conditions are present:

1. the processor is in either interrupt mode;
2. the proceed indicator is turned on;
3. a PDT instruction is extracted;

the A address of the PDT instruction is checked for violation as described on page E-2. If the effective A address references a protected memory location, the actions described below are performed.

1. When the violation occurs in the internal interrupt mode:
   a. The II address violation indicator is set.
   b. Further extraction of the instruction is not performed, and the contents of SR are advanced to the next sequential op code.
   c. An internal interrupt does not occur (since the processor is already in the internal interrupt mode). Rather, the condition of the II address violation indicator may be tested by the program (after the status of of the indicator is stored via an SVI instruction). The execution of the SVI instruction clears the indicator, so that the setting of this indicator does not eventually cause an internal interrupt.
2. When the violation occurs in the external interrupt mode:
   a. The EI address violation indicator is set.
   b. No further extraction is performed, and the contents of SR are advanced to the next sequential op code.
   c. An internal interrupt does not occur (since the processor is in the external interrupt mode). Rather, the condition of the EI address violation indicator can be tested after an SVI instruction is issued as described in 1.c., above.

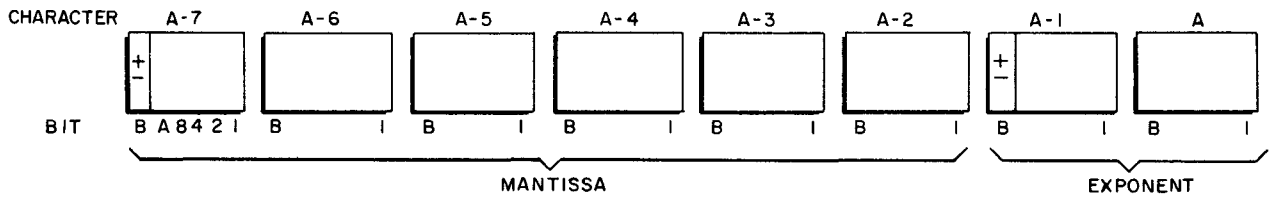
APPENDIX F

SCIENTIFIC UNIT FOR MODELS 1200 AND 2200

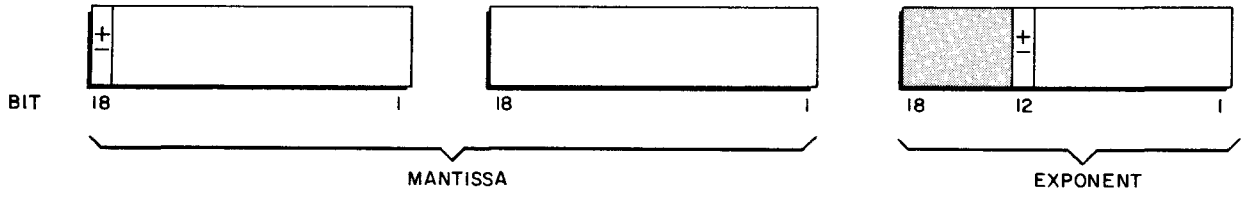
The scientific unit (Feature 1100) provides a repertoire of 12 floating-point instructions, a binary mantissa shift instruction, and a binary integer multiply instruction for the Type 1201 or 2201 processor. This appendix is a programmer's working summary of the hardware bulletin Scientific Unit for Models 1200 and 2200. Before referring to this appendix, the programmer should be familiar with the detailed functional and programming description of Feature 1100 presented in the information 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 $\pm 10^{\pm 616}$.



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.



FLOATING-POINT REGISTERS

The four addressable floating-point accumulators have the locations in control memory shown on page F-2.

Accumulator Address	Control Memory Location (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 cannot 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.

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

Pre-normalization: Mantissa of divisor is normalized (left-shifted) with adjusted exponent.

Equalization: Mantissa of operand with smaller exponent is right-shifted until exponents are equal.

Post-normalization: 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 character A through character 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.
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"
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.

TIMING NOTES

All timings shown are for Model 2200 and 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.

Table F-1. Summary of Scientific Instructions

FORMAT	OCTAL OP CODE	FUNCTION	REGISTERS AFTER OPERATION	TIMING[1]
DATA MOVING INSTRUCTIONS				
STORE FLOATING ACCUMULATOR				
FMA/A,X-,00 or TAM/A,X-	07	(X) is stored in (A). (X) is unaltered.	AAR: A-8 BAR: B_p	N_i + 10 cycles
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 cycles
Accumulator to accumulator: FAA/XY,02 or TAA/XY	06	(X) is loaded into Y. No normalization occurs.	AAR: A_p BAR: B_p	8 cycles
STORE LOW-ORDER RESULT				
Memory to accumulator: FMA/A,00,07 or TLM/A	07	(LOR) is stored in A. No normalization occurs.	AAR: A-8 BAR: B_p	N_i + 9 cycles
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 cycles
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 + 9 cycles
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 cycles

Table F-1 (cont). Summary of Scientific Instructions

FORMAT	OCTAL OP CODE	FUNCTION	REGISTERS AFTER OPERATION	TIMING[1]
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, post-normalization.	AAR: A-8 BAR: B_p LOR: Low-order result of operation. Sign bit = 0. Exponent = high-order exponent minus 35.	$N_i + 13 + \lceil N_n/4 \rceil$ cycles
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, post-normalization.	AAR: A_p BAR: B_p LOR: Low-order result of operation. Sign bit = 0. Exponent = high-order exponent minus 35.	$11 + \lceil N_n/4 \rceil$ cycles
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, post-normalization.	AAR: A-8 BAR: B_p LOR: Low-order result of operation. Sign bit = 0. Exponent = high-order exponent minus 35.	$N_i + 13 + \lceil N_n/4 \rceil$ cycles
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, post-normalization.	AAR: A_p BAR: B_p LOR: Low-order result of operation. Sign bit = 0. Exponent = high-order exponent minus 35.	$11 + \lceil N_n/4 \rceil$ cycles

Table F-1 (cont). Summary of Scientific Instructions

FORMAT		OCTAL OP CODE	FUNCTION	REGISTERS AFTER OPERATION	TIMING[1]
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: Post-normalization.	AAR: A-8 BAR: B_p LOR: Low-order product. Sign bit = 0. Exponent = high-order exponent minus 35.	$N_i + 21 + \lceil N_1/2 \rceil + \lceil N_n/4 \rceil$ cycles
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: Post-normalization.	AAR: A_p BAR: B_p LOR: Low-order product. Sign bit = 0. Exponent = high-order exponent minus 35.	$19 + \lceil N_1/2 \rceil + \lceil N_n/4 \rceil$ cycles
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: Pre-normalization (divisor), post-normalization quotient).	AAR: A-8 BAR: B_p LOR: Remainder. Sign = sign of dividend. Exponent = exponent of 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.	$N_i + 40 + \lceil N_n/4 \rceil$ cycles
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: Pre-normalization (divisor), post-normalization (quotient)	AAR: A_p BAR: B_p LOR: Remainder. Sign = sign of dividend. Exponent = exponent of 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.	$38 + \lceil N_n/4 \rceil$ cycles

Table F-1 (cont). Summary of Scientific Instructions

FORMAT	OCTAL OP CODE	FUNCTION	REGISTERS AFTER OPERATION	TIMING[1]
		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 + 24 cycles
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 + 23 cycles

Table F-1 (cont). Summary of Scientific Instructions

FORMAT	OCTAL OP CODE	FUNCTION	REGISTERS AFTER OPERATION	TIMING[1]
		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 NO BRANCH NXT BRANCH SR: NXT NO BRANCH JI(A) BRANCH	N_i + 4 cycles NO BRANCH N_i + 6 cycles BRANCH
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 + 4 cycles BRANCH N_i + 2 cycles NO BRANCH

Table F-1 (cont). Summary of Scientific Instructions

FORMAT	OCTAL OP CODE	FUNCTION	REGISTERS AFTER OPERATION	TIMING[1]
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 ($0 \leq V \leq 63$) specifies the number of positions by which bits are shifted. M=0 left, SP, rotate (end around) M=1 left, SP, arithmetic M=2 left, DP, rotate M=3 left, DP, arithmetic M=4 right, SP, rotate M=5 right, SP, arithmetic M=6 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	$9 + \lceil N_s/4 \rceil$ cycles
BINARY INTEGER ARITHMETIC INSTRUCTION				
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	$N_i + 20 + \lceil N_1/2 \rceil$ cycles

[1] All timings pertain to Model 2200 only.

HONEYWELL
ELECTRONIC
DATA
PROCESSING
WELLESLEY HILLS,
MASSACHUSETTS 02181