NCR REFERENCE MANUAL

An Educational Publication

PRODUCT INFORMATION--NCR CENTURY
PROCESSORS

number: 3
page: 1 of 91
date: Dec. 72

ST-9402-13
BINDER NO. 0141

NCR CENTURY 200 PROCESSOR

This publication contains the functional description of the NCR Century 200 processor. It is not intended as a reference manual for programming and operating the NCR Century 200.

TABLE OF CONTENTS

INTRODUCTION

ARITHMETIC LOGIC UNIT (CONT'D)

COMMANDS (CONT'D)

I/O CONTROL

OPTIONS

INTRODUCTION

THE NCR CENTURY 200 PROCESSOR

The NCR Century 200 processor, the principle component of the NCR Century 200 system, is a fast, flexible processing unit, which extends the range of the system from batch processing to online, real-time processing. Expandable memory and hardware capabilities, with numerous other optional features, facilitate system design to meet specific customer needs.

The NCR Century 200 has a repertoire of 39 hardware commands, including all the commands of the NCR Century 100. Commands that were executed by software simulation on the NCR Century 100 are part of the basic command set of the NCR Century 200. Optional commands make 315 and 1401 emulation/simulation, multiprogramming, and floating point arithmetic possible.

Addressing flexibility is extended by 63 index registers and four modes of addressing: direct addressing, two modes of indirect addressing, and incremental indexing of addresses. Data stored in memory is byte-addressable, which makes it possible to use variable-length fields.

Data may be represented in binary form, binary coded decimal form, or in NCR Century code, which conforms to the 8-bit ASCII code. Data fields may be signed or unsigned. Arithmetic operations are performed on either packed or unpacked data.

Extensive use is made of live, hardware registers in the NCR Century 200. Since there is virtually no time delay involved in accessing live registers, internal processing speed is substantially increased by their use. The use of live registers also eliminates the need for a specific clocking scheme, making the internal operation of the processor asynchronous, with memory accessed only when required for data retrieval or putaway.

The NCR Century 200 processor includes interrupt hardware which automatically directs the processor to an alternate execution path when necessary to respond to error conditions or normal interrupt conditions, such as I/O termination, tracing trap routine, etc.

THE NCR CENTURY 200 SYSTEM

The NCR Century 200 system establishes a flexible, economical base for expansion to provide the customer with a variety of processing and performance capabilities.

The basic system consists of the following: processor with 32K memory, I/O writer, integrated COT (card or tape reader), integrated printer, and dual spindle disc unit with controller. The I/O section of the basic system consists of four common trunks (quadraplex), with trunk 7 dedicated to the integrated printer and trunk 0 dedicated to the operator's console, the COT, and the I/O writer. The nominal data transfer rate of the common trunks is 120KB, with each trunk being able to accommodate 8 peripherals.

The basic system may be expanded with the optional features listed next.

Extended Memory

Memory sizes can be extended in variable increments up to 512K, as the customer's needs dictate. Within the memory sizes listed below, memory may be increased, with the necessary processor modifications, on the customer's premises.

The following memory sizes are available with the NCR Century 200:

32,768 bytes	98,304 bytes	262,144 bytes
49,152 bytes	131,072 bytes	393,216 bytes
65,536 bytes	196,608 bytes	524,208 bytes

Extended Bandwidth

The trunk bandwidth may be extended by converting trunk 6, or trunks 5 and 6, into high-speed trunks. Conversion to high-speed trunks increases the trunk transfer rate and, therefore, the bandwidth of the entire system. High-speed peripherals, which are those that have a transfer rate above 120KB, must be connected to a high-speed trunk. The system I/O bandwidth is extended from 320KB without high-speed trunks, to 487KB with one high-speed trunk, to 909KB with two high-speed trunks.

Extended I/O Simultaneity

A quadraplex system may be converted to an octaplex system by adding four 8-position trunks to the system and converting trunks 5 and 6 to high-speed trunks. The installation of this option extends the I/O capacity from 4-way simultaneity to 8-way simultaneity. The optional trunks (trunks 1, 2, 3, and 4) have a data transfer rate of 120KB. Trunks 5 and 6 must be converted to high-speed trunks with the addition of the four optional trunks; consequently, the system I/O bandwidth is also extended.

Multiprogramming

Multiprogramming is the concurrent execution of two or more programs. Stored in memory, under the control of a supervisor program (executive), are several programs that share processor and I/O facilities. Memory protection is provided by the base address register (BAR) and the limiting address register (LAR). BAR/LAR register contents can be changed only by the executive, to protect the integrity of the memory-resident programs (partitions). Each partition has its own set of 63 index registers, so that each program may assume a starting address of zero. Multiprogramming software provides a method of relocating programs in memory by the direct intervention and control of the operator. This assures efficient utilization of the available memory.

Any NCR Century 200 system may be modified to allow multiprogramming. Included in the multiprogramming option are a minimum of 64K memory, LOGIC and TABLE COMPARE commands, user/supervisor states of operation, BAR/LAR registers and LOAD BAR command, octaplex option, interval timer option, some additional functions by existing commands, additional control and compare logic functions, and additional console functions.

Floating Point Option

The floating point option automatically scales the numbers involved in a computation and maintains the precision of the result of the computation. The option comprises 12 commands that provide for addition, subtraction, comparison, multiplication, multiplication-addition, and division.

LOGIC and TABLE COMPARE Commands

The LOGIC command option permits logic operations, performed on Boolean algebra, to be performed on the individual bit positions of the two operands specified in the command. The function may be one of 16 logic functions, specified in the command.

The TABLE COMPARE command option makes it possible to decode two differently coded fields into a like format and then compare the two fields binarily, one byte at a time.

MULTIPLY Command

The MULTIPLY command option makes decimal multiplication of signed, packed fields possible.

Interval Timer

The interval timer option is a requirement in the multiprogramming environment, where the timer is used to prevent program looping and use of computer time by one partition in excess of the allocated time. The timer, which counts in one millisecond increments, is also intended to be used as a real-time clock (time-of-day) when used with the NCR BASIC.

Trace Option

The trace option provides the necessary hardware associated with program debugging. Each command being executed may be monitored by utilizing the trace permit flag, and three additional commands -- STORE TRACE, LOAD TRACE, LOAD MONITOR. The trace feature may be used with the Monitor switch on the console, which provides the means for checking a specific memory location. The specific memory location is loaded into the monitor register by program or from the console. With the Monitor switch in the ON position, whenever that specific location is written into, the trace permit flag is set and the trace routine is entered.

NCR 315 Compatibility

The NCR 315 emulator/simulator option enables the user to execute existing 315 programs and utility routines during the conversion process to the NCR Century 200 system. Simulation of the NCR 315 is accomplished by a freestanding emulator unit, the 315 Simulator Program, and three additional NCR Century 200 commands. The emulation unit contains the necessary logic for NCR 315 command execution, as well as the special BAR/LAR registers. The 315 Simulator Program executes by simulation those instructions that cannot be executed by the emulation unit. The three added commands necessary for emulation are EXECUTE,

PACK (Special), and UNPACK (Special). The trace option must be included in the NCR 315 emulation/simulation.

1401, 1440, 1240, and 1460 Compatibility

Compatibility between the NCR Century 200 and the 1400-series is provided by the 1400-series Simulator and Emulator programs. The Emulator program requires a freestanding emulation unit.

The 1400 Compatibility option permits execution of 1401, 1460, 1440, and 1240 instructions by simulation. Included in the option are five additional NCR Century 200 commands: 1401 SCAN, 1401 ADD, 1401 SUBTRACT, 1401 MOVE, and 1401 CONVERT. The LOGIC and TABLE COMPARE command option must be used with the 1400 Compatibility option.

If the 1400 Compatibility option includes the emulator unit, one more hardware command, in addition to the ones listed above, is included in the NCR Century 200 -- 1401 SETUP. The emulation unit is, in effect, a limited 1401 in the NCR Century 200. Instructions and data stored in memory are in 1401 format. Instruction setup is performed by the emulator unit and execution is performed by the simulator program. In addition to the LOGIC and TABLE COMPARE commands, the trace option must also be included if the emulator unit is included in the 1400 Compatibility option.

Thermal I/O Writer

The standard I/O writer may be substituted by an NCR Thermal Printer as the system I/O writer. Printing in the thermal printer is performed by heating selected elements in a single printhead and bringing it into light contact with heat-sensitive paper to form the character image. The main advantages of thermal printing in comparison to impact printing are reduction of noise and speed of operation. Since printing is done by heating selected elements (arranged in a 7x5 dot matrix) of the single printhead, the time required for positioning the cylindrical printhead mechanically is eliminated and the speed of the thermal printer is increased to five times that of the impact printer. Nominal data transmission rate of the thermal printer is 30 characters per second.

Remote I/O Writer

A second I/O writer, operating in parallel with the console I/O writer, may be added to the system. The remote I/O writer may be located up to 500 feet from the console. Being connected in parallel, both I/O writers print the input and output data simultaneously. If one of the I/O writers initiates an input, the other one may not initiate an input until the original input has terminated.

The console and the remote I/O writer must be of the same type, that is, a standard and a thermal I/O writer must not be mixed on the same system.

Operator Alarms

Three optional operator alarms are available: a software initiated alarm, a remote audible alarm, and an extra loud alarm. The software initiated alarm

provides the programmer with the ability to alert the operator that the system requires attention. The remote audible alarm works in parallel with the present console alarm, only difference being that the remote alarm may be located up to 100 feet from the console. The extra loud alarm is a separate unit on top of the power supply cabinet that draws power from a convenience outlet in the power supply. The primary purpose of the extra loud alarm is to provide a more effective operator alert system in a multi-system computer facility.

Peripheral Options

A great variety of printers and other peripherals are available on an optional basis.

The NCR 640-200, the integrated printer used in the basic NCR Century 200 configuration, may be substituted with an integrated 640-210 or 640-300 printer. If printing loads of the system so dictate, common trunk printers (with controllers) may be used in addition to or in place of the integrated printer. Printers vary according to character set (64 characters/single numeric or 52 characters/double numeric), print columns (132 or 160), and printing speeds (600/1200 lines per minute or 1500/3000 lines per minute).

Available with the 640-200 integrated printer is an optional SCRIBE/SPRINT type drum with 72 characters in the print set.

As a substitute for the NCR 655-201 disc unit and controller, the user may select the optional NCR 657-101/102 disc unit with controller, which not only increases the total storage capacity of the magnetic disc file, but also increases the transfer rate of the data being read from or written on the disc and, therefore, the entire system throughput.

Optional peripheral units are too numerous to mention in this publication since its primary purpose is the functional description of the NCR Century 200 processor. For more information on peripheral units, see the individual unit descriptions in this binder.

PHYSICAL DESCRIPTION

The NCR Century 200 utilizes both short-rod and core memories. The basic storage element of the short-rod memory is a metallic rod coated with a thin magnetic film. The rod may be magnetized in two different directions, providing two stable states to represent a binary 0 or a binary 1.

The basic storage element of the core memory is a ferrite core, which may be magnetized in two different directions to provide two stable states that represent a binary 0 or a binary 1.

The basic data unit stored in memory is an 8-bit byte, plus an internally generated parity bit. The parity bit is generated and stored in memory with each data byte. When data is read from memory, the parity bit is checked to ensure the validity of the data bits. The parity bit is used only internally and is not taken into consideration in programming or data preparation for input to the processor.

The memory of the NCR Century 200 is expandable from 32K to 512K in the following increments:

32,768 bytes (32K)	98,304 bytes (96K)	262,144 bytes (256K)
49,152 bytes (48K)	131,072 bytes (128K)	393,216 bytes (384K)
65,536 bytes (64K)	196,608 bytes (192K)	524,288 bytes (512K)

Within the increment limitations shown above, a smaller memory may be up-graded to a larger memory on the customer's premises.

DATA REPRESENTATION

The 8 bits in a byte may be used to represent two packed binary coded decimal (BCD) numbers, 8-bit binary numbers, or 8-bit NCR Century characters (4 zone bits and 4 digit bits).

NCR CENTURY CODE CHART

B_8-B_5 ↓ \ B_4-B_1 →		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
0001	1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0010	2	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0011	3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0100	4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0101	5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0110	6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0111	7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Note that in the code chart b8 is always 0, limiting the number of possible characters to 128. This configuration conforms to the American Standard Code for Information Interchange (ASCII).

Examples of Data Representation

b8	b7	b6	b5	b4	b3	b2	b1
0	1	0	1	0	1	1	0

If the bit configuration above is considered as two 4-bit BCD numbers, the decimal values are 5 and 6.

8	4	2	1	8	4	2	1
0	1	0	1	0	1	1	0

position values

    5   6

If the same bit configuration is considered as a single, 8-bit binary number, the decimal value is 86.

128	64	32	16	8	4	2	1
0	1	0	1	0	1	1	0

position values

            86

If the same bit configuration is considered as a character in the NCR Century code, it is equivalent to a V.

b8	b7	b6	b5	b4	b3	b2	b1
0	1	0	1	0	1	1	0

    zone   digit
    bits   bits

MEMORY ADDRESSING

Character locations in memory are numbered consecutively, starting at 0; each number is the address of one byte. A group of contiguous bytes is called a field and is addressed by the leftmost byte (most significant character). All hardware-recognized commands perform an addressing function to access a memory field. The contents of this field are the data upon which the command performs its operation. Commands can operate upon fields ranging in size from 1 through 256 bytes (the MULTIPLY command is an exception). Attempting to access a location greater than memory size causes a program error (PE) condition.

In the following example, the 5 character field (A) containing the data 42385, is referenced by the address 100. If the same data were divided into two fields (B and C) the address of field B, containing the data 423, would be 100, and the address of field C, containing the data 85, would be 103.

FIELD A

ADDRESS	100	101	102	103	104
CONTENTS	4	2	3	8	5

FIELD B   FIELD C

ADDRESS	100	101	102	103	104
CONTENTS	4	2	3	8	5

For simplicity, the addresses in the example are given in decimal form. Actual memory addresses are in binary form, consisting of 16 bits (two bytes) for memory sizes up to and including 65,536 characters and 19 bits for systems with the extended memory option.

To simplify address entry through the operator's console, the hexadecimal notation system is used. Hexadecimal numbers are expressed to the base 16 and are related to binary and to decimal numbers as shown in this table:

CONVERSION TABLE		
DECIMAL	BINARY	HEXADECIMAL
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

To convert from binary to hexadecimal, separate the binary address into four 4-bit groups and assign the appropriate hexadecimal digit to each group:

$$0011010111101010 = \underbrace{0011}_{3} \quad \underbrace{0101}_{5} \quad \underbrace{1110}_{E} \quad \underbrace{1010}_{A} = 35EA$$

To convert from hexadecimal to binary, reverse the procedure:

$$2F9A = \underset{0010}{2} \quad \underset{1111}{F} \quad \underset{1001}{9} \quad \underset{1010}{A} = 0010111110011010$$

Should it become necessary to convert from hexadecimal to decimal, each position of the hexadecimal number must be expanded as follows:

$$34F6 = (3 \times 16^3) + (4 \times 16^2) + (15 \times 16^1) + (6 \times 16^0)$$

$$= (3 \times 4{,}096) + (4 \times 256) + (15 \times 16) + (6 \times 1)$$

$$= (12{,}288 + 1024 + 240 + 6)$$

$$= 13{,}558$$

Hexadecimal/decimal conversion can be simplified by using the following table:

HEXADECIMAL/DECIMAL CONVERSION					
HEXADECIMAL EQUIVALENT	POSITION 5*	POSITION 4	POSITION 3	POSITION 2	POSITION 1
0	0	0	0	0	0
1	65,536	4,096	256	16	1
2	131,072	8,192	512	32	2
3	196,608	12,288	768	48	3
4	262,144	16,384	1,024	64	4
5	327,680	20,480	1,280	80	5
6	393,216	24,576	1,536	96	6
7	458,752	28,672	1,792	112	7
8		32,768	2,048	128	8
9		36,864	2,304	144	9
A		40,960	2,560	160	10
B		45,056	2,816	176	11
C		49,152	3,072	192	12
D		53,248	3,328	208	13
E		57,344	3,584	224	14
F		61,440	3,840	240	15
*Position 5 is used with systems having the extended memory option.					

To convert from a hexadecimal address to the equivalent decimal address, select the decimal value from each of the four columns that corresponds to the hexadecimal digit and add the values:

	Position 4	Position 3	Position 2	Position 1
3FOB =	3	F	0	B

$$= 12{,}288 + 3{,}840 + 0 + 11 = 16{,}139$$

To convert from a decimal address to a hexadecimal address, a series of subtractions is used:

Decimal Address = 16,139

1. Locate the decimal value in the most-significant position (position 4)* that is equal to or less than the decimal address. Subtract this value from the decimal address.

$$\begin{array}{r} 16139 \\ -12288 \\ \hline 3851 \end{array} = 3 \text{ (Position 4)}$$

2. From the next position, select the decimal value that is equal to or less than the result of the subtraction and subtract this number from that result.

$$\begin{array}{r} 3851 \\ -3840 \\ \hline 11 \end{array} = F \text{ (Position 3)}$$

3. Repeat Step 2 for the remaining positions.

$$\begin{array}{r} 11 \\ - \quad 0 \\ \hline 11 \end{array} = 0 \text{ (Position 2)}$$

*Position 5 in systems having extended memory option.

$$\begin{array}{r} - \quad 11 \\ \hline 0 \end{array} = B \text{ (Position 1)}$$

Hexadecimal Address = 3FOB

INDEX REGISTERS

The NCR Century 200 memory contains 63 index registers, designated IR1 through IR63. The optional use of index registers provides flexibility in addressing memory by allowing the storage of a constant that can be used to point to the beginning of a memory field. The partial address portion of the command can then be used to give an offset from the beginning of the memory field, allowing addressing of any byte or group of bytes within the field. (Refer to the Arithmetic Logic Unit section.) Each index register is located in the 3 low-order (rightmost) bytes of a four-byte index register word. Only the 2 rightmost bytes are used in addressing memories of 65,536 characters or less. Systems containing the extended memory option (memory sizes larger than 65,536 characters) require, in addition to the two rightmost bytes, the use of the three low-order bits of the third byte for addressing. These extra three bits increase the binary field from 16 bits to 19 bits, which allows a maximum binary value of 524,287, capable of addressing the NCR Century 200's largest memory. Index words are located consecutively in memory locations 0004 through 00FF (4 through 255) as shown in the following illustration of index register and reserved memory areas.

INDEX REGISTERS AND RESERVED MEMORY AREAS

Section	Index Register Word	IR	Decimal Address	B8	B7	B6	B5	B4	B3	B2	B1	Hex Address
RESERVED REGS			00000	NOT USED BY NCR CENTURY 200								0000
			00001	NOT USED BY NCR CENTURY 200								0001
			00002	NOT USED BY NCR CENTURY 200								0002
			00003	NOT USED BY NCR CENTURY 200								0003
	INDEX REGISTER WORDS 1-7		00004	RL	RL	RL	RL	RL	RL	RL	RL	0004
			00019	RL	RL	RL	RL	RL	RL	RL	RL	0013
ERROR STATUS WORD	INDEX REGISTER WORD 5	IR5	00020	Q	Q	Q	Q	Q	Q	Q	Q	0014
			00021	NOT USED BY NCR CENTURY 200					A_3	A_3	A_3	0015
			00022	A_2	A_2	A_2	A_2	A_2	A_2	A_2	A_2	0016
			00023	A_1	A_1	A_1	A_1	A_1	A_1	A_1	A_1	0017
	INDEX REGISTER WORD 6	IR6	00024	T	T	T	T	T	T	T	T	0018
			00025	NOT USED BY NCR CENTURY 200					B_3	B_3	B_3	0019
			00026	B_2	B_2	B_2	B_2	B_2	B_2	B_2	B_2	001A
			00027	B_1	B_1	B_1	B_1	B_1	B_1	B_1	B_1	001B
	INDEX REGISTER WORD 7	IR7	00028	S		OF	RI	TP	G	E	L	001C
			00029	NOT USED BY NCR CENTURY 200					CR_3	CR_3	CR_3	001D
			00030	CR_2	CR_2	CR_2	CR_2	CR_2	CR_2	CR_2	CR_2	001E
			00031	CR_1	CR_1	CR_1	CR_1	CR_1	CR_1	CR_1	CR_1	001F
SPECIAL REGISTERS	INDEX REGISTER WORD 8	IR8	00032	RC	RC	RC	RC	RC	RC	RC	RC	0020
			00033	NOT USED BY NCR CENTURY 200					L3	L3	L3	0021
			00034	L_2	L_2	L_2	L_2	L_2	L_2	L_2	L_2	0022
			00035	L_1	L_1	L_1	L_1	L_1	L_1	L_1	L_1	0023
	INDEX REGISTER WORD 9	IR9	00036	EC	EC	EC	EC	EC	EC	EC	EC	0024
			00037	NOT USED BY NCR CENTURY 200					N_3	N_3	N_3	0025
			00038	N_2	N_2	N_2	N_2	N_2	N_2	N_2	N_2	0026
			00039	N_1	N_1	N_1	N_1	N_1	N_1	N_1	N_1	0027
PROGRAM STATUS WORDS	INDEX REGISTER WORD 10	IR10	00040	Q	Q	Q	Q	Q	Q	Q	Q	0028
			00041	NOT USED BY NCR CENTURY 200					A_3	A_3	A_3	0029
			00042	A_2	A_2	A_2	A_2	A_2	A_2	A_2	A_2	002A
			00043	A_1	A_1	A_1	A_1	A_1	A_1	A_1	A_1	002B
	INDEX REGISTER WORD 11	IR11	00044	T	T	T	T	T	T	T	T	002C
			00045	NOT USED BY NCR CENTURY 200					B_3	B_3	B_3	002D
			00046	B_2	B_2	B_2	B_2	B_2	B_2	B_2	B_2	002E
			00047	B_1	B_1	B_1	B_1	B_1	B_1	B_1	B_1	002F
	INDEX REGISTER WORD 12	IR12	00048	S		OF	RI	TP	G	E	L	0030
			00049	NOT USED BY NCR CENTURY 200					CR_3	CR_3	CR_3	0031
			00050	CR_2	CR_2	CR_2	CR_2	CR_2	CR_2	CR_2	CR_2	0032
			00051	CR_1	CR_1	CR_1	CR_1	CR_1	CR_1	CR_1	CR_1	0033
TRACE INTERRUPT	INDEX REGISTER WORD 13	IR13	00052	Q	Q	Q	Q	Q	Q	Q	Q	0034
			00053	NOT USED BY NCR CENTURY 200					A_3	A_3	A_3	0035
			00054	A_2	A_2	A_2	A_2	A_2	A_2	A_2	A_2	0036
			00055	A_1	A_1	A_1	A_1	A_1	A_1	A_1	A_1	0037
PROGRAM INTERRUPT	INDEX REGISTER WORD 14	IR14	00056	T	T	T	T	T	T	T	T	0038
			00057	NOT USED BY NCR CENTURY 200					B_3	B_3	B_3	0039
			00058	B_2	B_2	B_2	B_2	B_2	B_2	B_2	B_2	003A
			00059	B_1	B_1	B_1	B_1	B_1	B_1	B_1	B_1	003B
OPTION	INDEX REGISTER WORD 15	IR15	00060	S		OF	RI	TP	G	E	L	003C
			00061	NOT USED BY NCR CENTURY 200					CR_3	CR_3	CR_3	003D
			00062	CR_2	CR_2	CR_2	CR_2	CR_2	CR_2	CR_2	CR_2	003E
			00063	CR_1	CR_1	CR_1	CR_1	CR_1	CR_1	CR_1	CR_1	003F
WORK REGS	INDEX REGISTER WORD 16 - 63		00064	CC	CC	CC	CC	CC	CC	CC	CC	0040
			00065	X	X	X	X	X	X	X	X	0041

KEY

Symbol		Description
RL	=	RESERVED REGISTERS USED BY SOFTWARE TO PERMIT UPWARD COMPATIBILITY FROM NCR CENTURY 100
Q	=	COMMAND CODE
A_2A_1	=	EFFECTIVE A ADDRESS
A_3	=	USED WITH A_2A_1 WHEN MEMORY LARGER THAN 65,536 BYTES
T	=	FIELD LENGTH
B_2B_1	=	EFFECTIVE B ADDRESS
B_3	=	USED WITH B_2B_1 WHEN MEMORY LARGER THAN 65,536 BYTES
S	=	S FLAG (MULTIPROGRAMMING OPTION)
OF	=	OVERFLOW FLAG
RI	=	REPEAT INDICATOR
TP	=	TRACE PERMIT (TRACE OPTION)
G	=	"GREATER" FLAG
E	=	"EQUAL" FLAG
L	=	"LESS" FLAG
CR_2CR_1	=	CONTROL REGISTER
CR_3	=	USED WITH CR_2CR_1 WHEN MEMORY LARGER THAN 65,536 BYTES
RC	=	REPEAT COUNTER
L_2L_1	=	RETURN LINK STORED UPON EXECUTION OF JUMP COMMAND
L_3	=	USED WITH L_2L_1 WHEN MEMORY LARGER THAN 65,536 BYTES
EC	=	ERROR CODE STORAGE

TYPE OF ERROR

1	=	PE
2	=	TRAPPING COMMAND CODE (NOT PRIVILEGED)
3	=	PE AND TRAPPING COMMAND CODE (NOT PRIVILEGED)
5	=	PE CAUSED BY CHARACTERISTIC OVERFLOW IN SYSTEMS WITH THE FLOATING POINT OPTION.
8	=	TRAPPING COMMAND CODE CAUSED BY PRIVILEGED COMMAND OCCURRING IN USER STATE - MULTIPROGRAMMING OPTION
9	=	PE AND TRAPPING COMMAND CODE CAUSED BY PRIVILEGED COMMAND OCCURRING IN THE USER STATE

Symbol		Description
N_2N_1	=	NEXT ADDRESS STORED BY THE DECODE AND SCAN COMMANDS
N_3	=	USED WITH N_2N_1 WHEN MEMORY LARGER THAN 65,536 BYTES
CC	=	COUNT COUNTER (USED BY COUNT COMMAND)
X	=	MISCELLANEOUS INDEX REGISTER WORDS
ME_2ME_1	=	ADDRESS TO WHICH CONTROL IS TRANSFERRED WHEN A MEMORY ERROR TRAP OCCURS
ME_3	=	USED WITH ME_2ME_1 WHEN MEMORY LARGER THAN 65,536 BYTES
PE_2PE_1	=	ADDRESS TO WHICH CONTROL IS TRANSFERRED WHEN A PROGRAMMING ERROR, TRAPPING COMMAND CODE, OR BOTH OCCUR.
PE_3	=	USED WITH PE_2PE_1 WHEN MEMORY LARGER THAN 65,536 BYTES
I_2I_1	=	ADDRESS TO WHICH CONTROL IS TRANSFERRED WHEN PROGRAM INTERRUPT OCCURS.
I_3	=	USED WITH I_2I_1 WHEN MEMORY LARGER THAN 65,536 BYTES
TC_2TC_1	=	ADDRESS TO WHICH CONTROL IS TRANSFERRED WHEN EITHER A TRACING OR AN ADDRESS MONITOR TRAP OCCURS WHEN THE TRACE OPTION IS INCLUDED
TC_3	=	USED WITH TC_2TC_1 WHEN MEMORY LARGER THAN 65,536 BYTES

INDEX REGISTERS AND RESERVED MEMORY AREAS [CONT.]

	DECIMAL ADDRESS	B8	B7	B6	B5	B4	B3	B2	B1	HEX ADDRESS	KEY
ME CONTROL AREA	00256									0100	
	00257	NOT USED BY NCR CENTURY 200					ME_3	ME_3	ME_3	0101	
	00258	ME_2	ME_2	ME_2	ME_2	ME_2	ME_2	ME_2	ME_2	0102	
	00259	ME_1	ME_1	ME_1	ME_1	ME_1	ME_1	ME_1	ME_1	0103	
PE CONTROL AREA	00260									0104	
	00261	NOT USED BY NCR CENTURY 200					PE_3	PE_3	PE_3	0105	
	00262	PE_2	PE_2	PE_2	PE_2	PE_2	PE_2	PE_2	PE_2	0106	
	00263	PE_1	PE_1	PE_1	PE_1	PE_1	PE_1	PE_1	PE_1	0107	
	00264	NOT USED BY NCR CENTURY 200								0108	
	00271	NOT USED BY NCR CENTURY 200								010F	
INTERRUPT CONTROL AREA (ICA)	00272									0110	
	00273	NOT USED BY NCR CENTURY 200					I_3	I_3	I_3	0111	
	00274	I_2	I_2	I_2	I_2	I_2	I_2	I_2	I_2	0112	
	00275	I_1	I_1	I_1	I_1	I_1	I_1	I_1	I_1	0113	
TRANSACTION COUNT CONTROL	00276									0114	
	00277	NOT USED BY NCR CENTURY 200					TC_3	TC_3	TC_3	0115	
	00278	TC_2	TC_2	TC_2	TC_2	TC_2	TC_2	TC_2	TC_2	0116	
	00279	TC_1	TC_1	TC_1	TC_1	TC_1	TC_1	TC_1	TC_1	0117	

LOCATIONS: 280 — 283   (0118 — 011B)   AND

288 — 351   (0120 — 012D)   AND

1024 — 3071   (0400 — 0BFF)   AND

6144 — 6355   (1800 — 18D3)

ARE RESERVED FOR OPTIONAL FEATURES AND I/O CONTROL

LOCATIONS: 284 — 287   (011C — 011F)

ARE NOT USED

Note that IR1 through IR15 are reserved for hardware and software operations within the processor. Except for multiprogramming systems, there is no hardware protection provided for index registers. This means that the programmer must never access these locations indiscriminately, since their contents can be altered.

FUNCTIONAL DESCRIPTION

Introduction

The function of the NCR Century memory unit is to receive information from the processor (ALU or I/O Control), to retain it, and to return it to the processor upon request. Communication between processor and memory falls into four general categories: control, timing, addressing, and information transfer.

```
┌─────────────────────────────────────────────────────┐
│        MEMORY/PROCESSOR COMMUNICATION               │
├─────────────────────────────────────────────────────┤
│  ┌─────┐          CONTROL          ────►  ┌─────┐   │
│  │  P  │                                   │     │   │
│  │  R  │  ◄────    TIMING                  │  M  │   │
│  │  O  │                                   │  E  │   │
│  │  C  │          ADDRESSING       ────►   │  M  │   │
│  │  E  │                                   │  O  │   │
│  │  S  │  ◄────   INFORMATION      ────►   │  R  │   │
│  │  S  │                                   │  Y  │   │
│  │  O  │                                   │     │   │
│  │  R  │                                   │     │   │
│  └─────┘                                   └─────┘   │
└─────────────────────────────────────────────────────┘
```
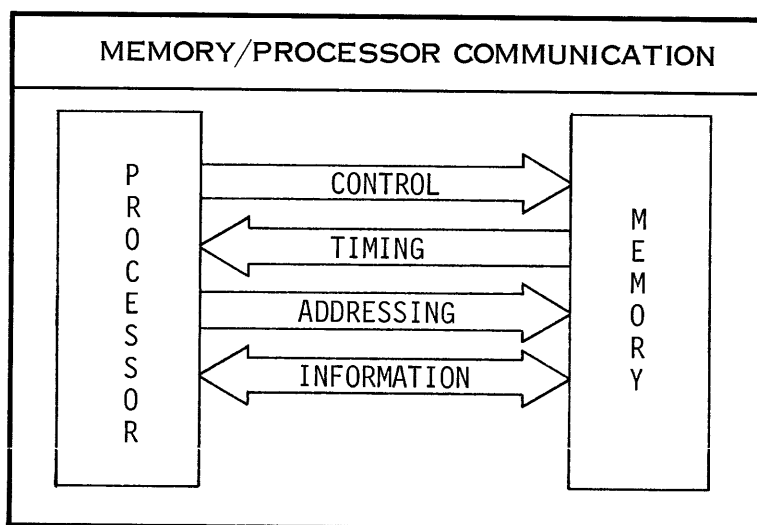
● Control

   Control from the processor performs two basic functions: initiation of the timing sequence, and request for a read or write operation.

● Timing

   Processor timing is controlled by hardware-generated signals from memory, which occur at specific intervals during the memory cycle.

● Addressing

   Addressing from the processor to memory takes place over 16 lines (19 lines if system has extended memory option). The address on these lines is decoded by memory hardware to provide character (byte) addressability.

● Information Transfer

Information (commands or data) is transferred between the processor and memory 2 bytes at a time, including the parity bits.

Functional Operation

The NCR Century 200 memory performs two basic operations, a read/restore operation and a clear/write operation, each requiring one memory cycle. The processor informs memory during one cycle that a read or write operation will be performed during the following cycle. As soon as this happens, hardware initiates a timing sequence that divides the memory cycle into timing signals that govern the remainder of the operation. At the beginning of the cycle, the memory address register accepts an address from the processor, decodes it, and selects the proper location in the memory storage area. During the first part of a read/restore operation, 1 bits detected in the selected area are transferred to the data register where the bytes are assembled and transmitted to the processor. During the second half of the operation, the information that was read from memory is restored to its original memory location.

A write operation also consists of two functions performed during a single memory cycle. Once the area has been selected, it is cleared so that all bits are 0. During this time, the processor has transmitted the data to be written into the data register. During the second half of the cycle, the data is written into the selected area by setting the appropriate bits to 1.

MEMORY/PROCESSOR COMMUNICATION
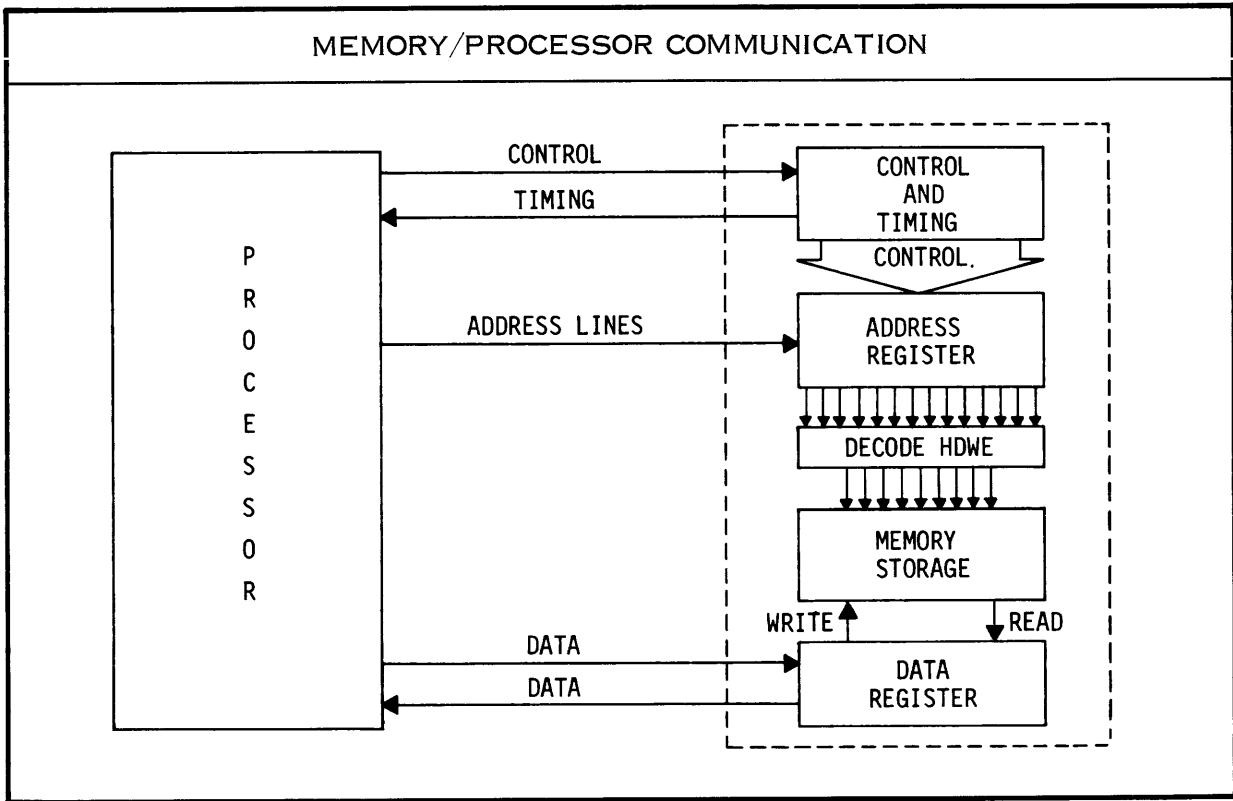
INTRODUCTION

The functional description of the NCR Century Arithmetic Logic Unit (ALU) operation contained in this section, which is primarily hardware-oriented, assumes that a program is resident in memory and being executed by the processor.

COMMANDS

Object program commands are stored in memory in 1-address or 2-address formats, occupying 4 or 8 bytes. Even though most commands require two operand addresses (an add command, for example, obviously requires the addresses of the two operands to be added), the two formats are functionally equivalent since the 1-address format uses the B operand address from the preceding command. In general, the results of an arithmetic operation replace the contents of the original B operand address.

The address of the leftmost character (Q) of each command must be evenly divisible by 4. An attempt to execute a command that violates this rule results in a program error (PE) interruption.

CENTURY HARDWARE COMMAND FORMAT

TWO-ADDRESS

ONE-ADDRESS

Q	RA	A2	A1	T	RB	B2	B1

b8-b3
Index
Register

Partial Address
of A Operand

Field
Length

(See RA)

Partial Address
of B Operand

b7-b1
Command
Code

b2-b1 Mode of
Addressing
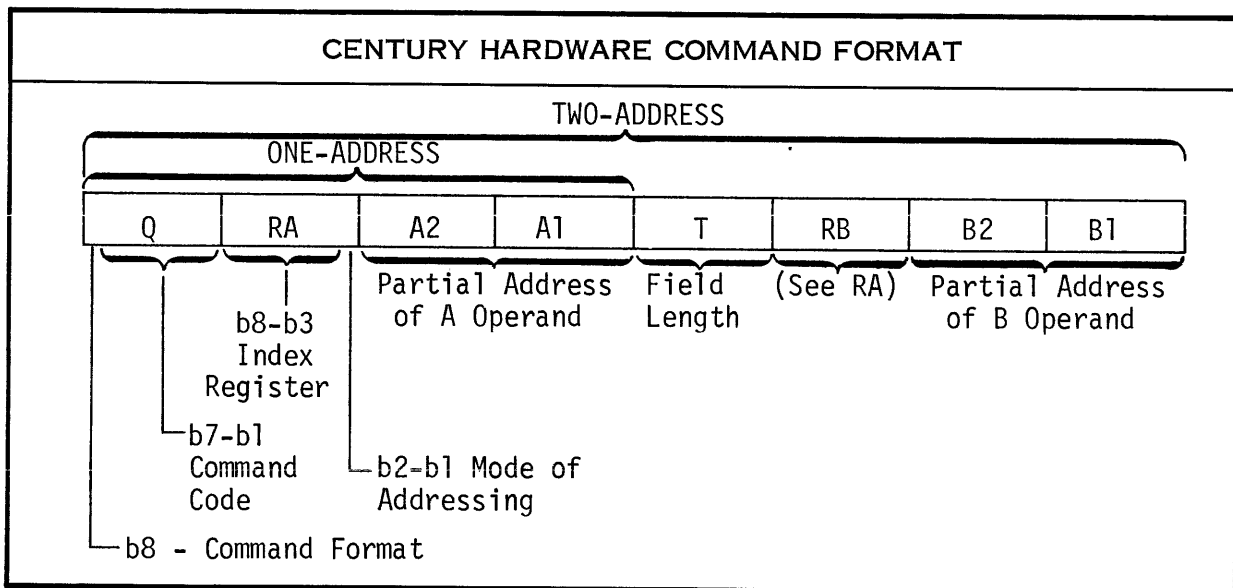
b8 - Command Format

Command Code - Q

The Q portion of the command specifies the command code and the command format. The binary value of $b7 - b1$ designates the command to be executed (add, subtract, etc.). The most significant bit (b8) indicates the command format:

- o b8 = 0 The command is a 2-address format.
- o b8 = 1 The command is a 1-address format; the address of the B operand and length character (T), if required, must be obtained from the preceding command.

Index Register - RA

The NCR Century 200 provides four modes of addressing and the optional use of index registers with each mode. The use of index registers allows greater flexibility for addressing. The RA character of the command specifies whether indexing is to be performed, the index register to be used, and which of four modes of addressing is required.

The binary value of the $b2 - b1$ portion of the RA character specifies the mode of addressing.

- $b2 - b1 = 0$ Mode 0, Direct addressing
- $b2 - b1 = 1$ Mode 1, Indirect addressing
- $b2 - b1 = 2$ Mode 2, Indirect addressing
- $b2 - b1 = 3$ Mode 3, Incremental indexing

If $b8 - b3 = 0$, no indexing is specified and the partial address (A2A1 portion) becomes the effective address. If modes 1 or 2 are used, the A2A1 portion becomes the effective indirect address.

If $b8 - B3 \neq 0$, indexing is specified and the binary value of these bits locates one of 63 index registers (IR1 - IR63). The partial address is then used to modify the contents of the index register and derive the effective address.

- Mode 0

    If $b8 - b3 = 0$, no index register is specified and the A2A1 characters form the effective address.

    If $b8 - B3 \neq 0$, the A2A1 characters are added to the contents of the specified index register to form the effective address; the contents of the specified index register remain unchanged.

    EXAMPLE 1:

    Mode 0   No indexing (Addresses are shown in decimal notation for simplification.)

COMMAND

10000	10001	10002	10003
Q	RA	A2	A1
	0	0	29750

MODE

EFFECTIVE ADDRESS OF A OPERAND

NO INDEX
REGISTER
SPECIFIED
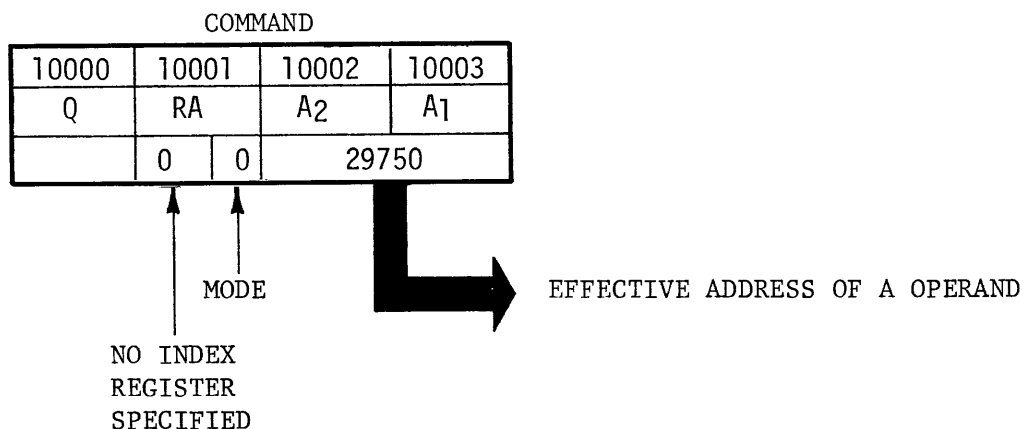
EXAMPLE 2:

Mode 0 Indexing

COMMAND

10000	10001	10002	10003
Q	RA	A$_2$	A$_1$
	19	0	14723

MODE

DECIMAL
ADDRESS

0076	0077	0078	0079
			3274

INDEX REGISTER WORD 19

14723	PARTIAL ADDRESS
+ 3274	CONTENTS OF IR19
17997	EFFECTIVE ADDRESS OF A OPERAND
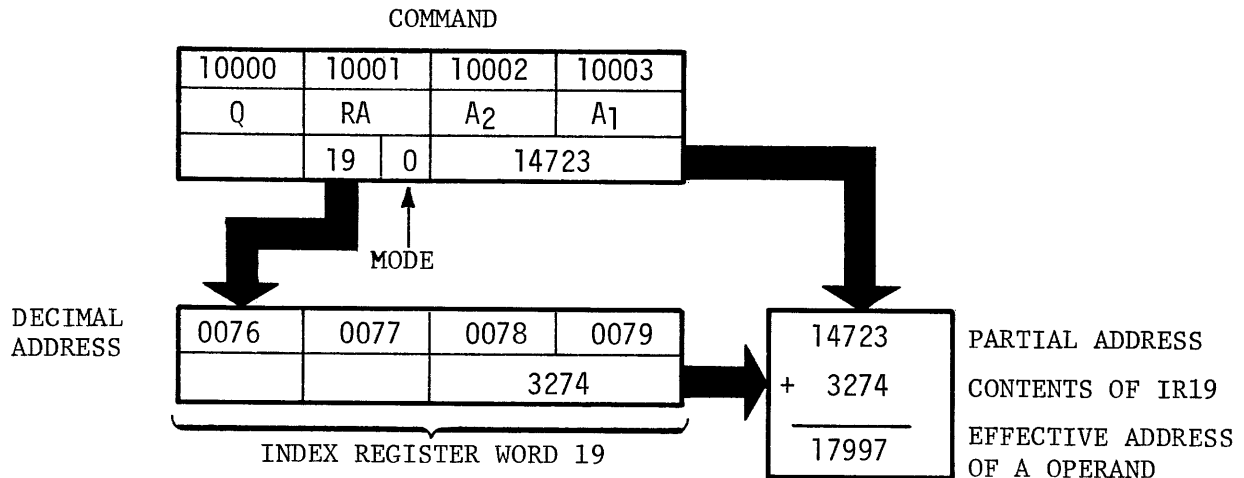
NOTE

The program specifies index register 19 (decimal equivalent
address = 0076) but the processor reads out addresses 0078
and 0079 for the contents of the index register. Refer to
Memory section of this publication for detailed description
of index registers.

● Mode 1

If b8 − b3 = 0, no index register is specified and the A2A1 characters form
an effective indirect address.

If b8 − b3 ≠ 0, the A2A1 characters are added to the contents of the speci-
fied index register to form an effective indirect address.

The contents of the 4-character field specified by the intermediate effec-
tive address are read out of memory and used to form another intermediate
effective address provided there is no mode change at the newest address.
Mode 1 addressing can be repeated a maximum of five times; that is, five
intermediate effective addresses can be used without a mode change. A PE
occurs on the sixth repeat if no mode change is initiated. The mode 1 ad-
dressing flow is repeated for each intermediate field referenced; there-
fore, the fields addressed must be at locations that are evenly divisible
by 4 or a PE results.

A change to Mode 0, 2, or 3 may be initiated at any of the intermediate
fields referenced. Whichever mode is specified, the rule governing that
particular mode is followed until the addressing flow is complete.

EXAMPLE:

Mode 1   No Indexing and Indexing

COMMAND

10000	10001	10002	10003
Q	RA	A$_2$	A$_1$
	0	1	16384

EFFECTIVE INDIRECT ADDRESS

↑ ↑
MODE

NO INDEX
REGISTER
SPECIFIED

16384	16385	16386	16387
	23	1	14216

↑
MODE

0092	0093	0094	0095
			18432

INDEX REGISTER WORD 23

```
  14216
+18432
------
 32648
```

PARTIAL ADDRESS

CONTENTS OF IR23

EFFECTIVE INDIRECT
ADDRESS

32648	32649	32650	32651
		X	
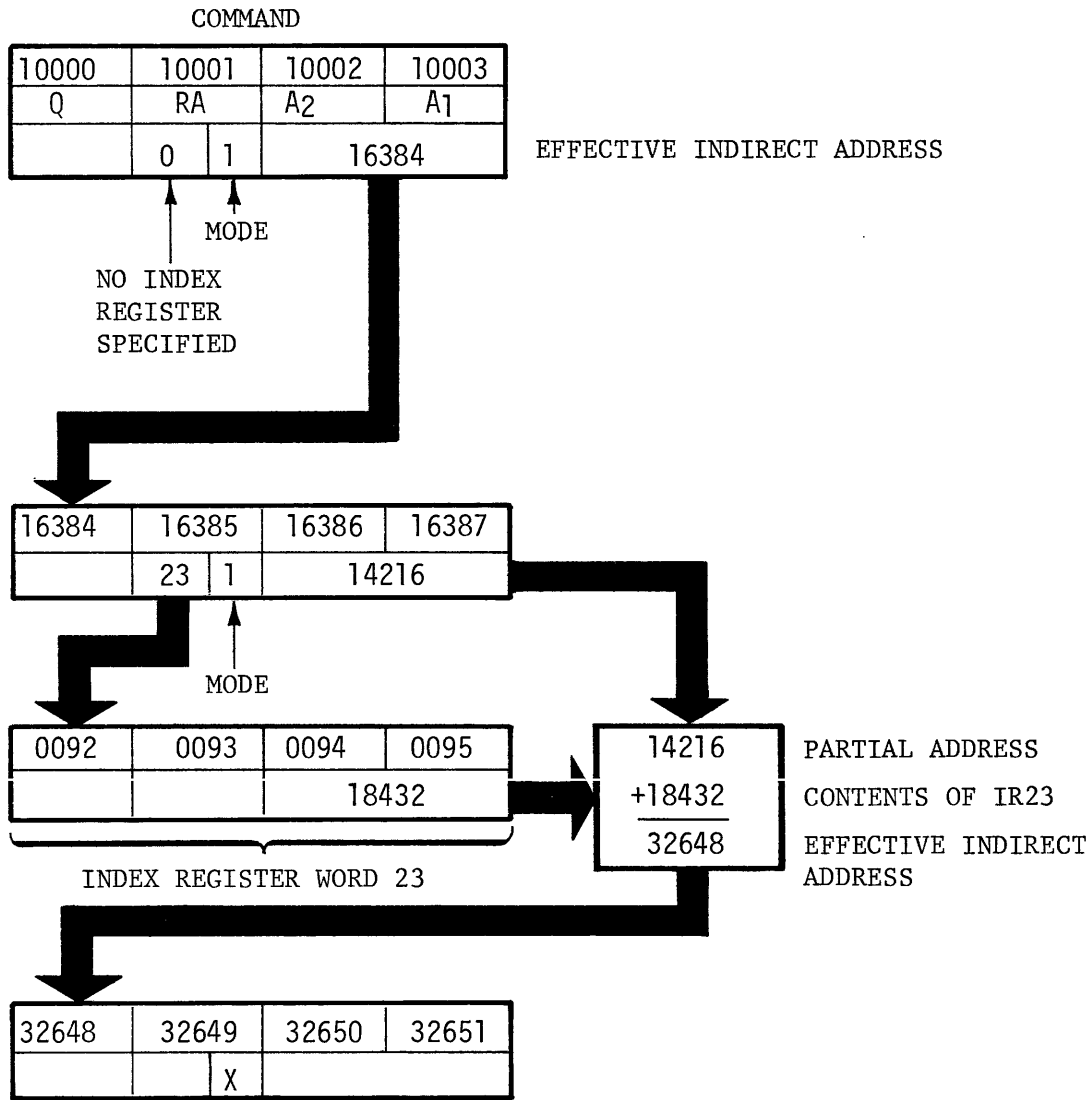
↑
MODE — May remain mode 1 for a maximum of five intermediate
addresses.

<u>NOTE</u>

If, in mode 1 indirect addressing, an intermediate address
results in an ME or PE, a change to mode 3 indexing during
the same command setup period is inhibited.

- **Mode 2**

  If b8 – b3 = 0, no index register is specified and the A2A1 characters form an effective indirect address.

  If b8 – b3 ≠ 0, the A2A1 characters are added to the contents of the specified index register to form an effective indirect address; the contents of the specified index register remain unchanged.

  The effective indirect address, which need not be evenly divisible by 4, locates a three character field whose contents function as the effective address for that operand.

  EXAMPLE 1:

  Mode 2   No Indexing

  COMMAND

10000	10001	10002	10003
Q	RA	A2	A1
	0 2	8507

  EFFECTIVE INDIRECT ADDRESS

  ↑ MODE
  NO INDEX
  REGISTER
  SPECIFIED

8507	8508	8509
	24790

  EFFECTIVE ADDRESS OF A OPERAND

  └─ b3–b1 Provides addressing for extended memory.

  Mode 2   Indexing

10000	10001	10002	10003
Q	RA	A2	A1
	60 2	10041

  ↑ MODE

0240	0241	0242	0243
		5001

  INDEX REGISTER WORD 60

  | 10041 |
  | + 5001 |
  | 15042 |

  PARTIAL ADDRESS

  CONTENTS OF IR60

  EFFECTIVE INDIRECT
  ADDRESS

15042	15043	15044
	8192

  EFFECTIVE ADDRESS OF A OPERAND

  └─ b3–b1 Same as above.

● Mode 3

If b8 - b3 = 0, no incremental indexing is performed and the A2A1 characters form the effective address.

If b8 - b3 ≠ 0, the A2A1 characters are added to the contents of the specified index register to form the effective address. This sum is stored back in the designated index register.

When command setup terminates, the address in the index register is equal to the effective A operand address. This technique is used to step through tables.

EXAMPLE 1:

Mode 3   Indexing

COMMAND

Q	RA	A₂	A₁
	45	3	19000

↑ MODE

0180	0181	0182	0183
			4640

INDEX REGISTER WORD 45
BEFORE SETUP

```
  19000      PARTIAL ADDRESS
+ 4640       CONTENTS OF IR45
──────       EFFECTIVE ADDRESS
  23640      OF A OPERAND
```

0180	0181	0182	0183
			23640

INDEX REGISTER WORD 45
AFTER SETUP
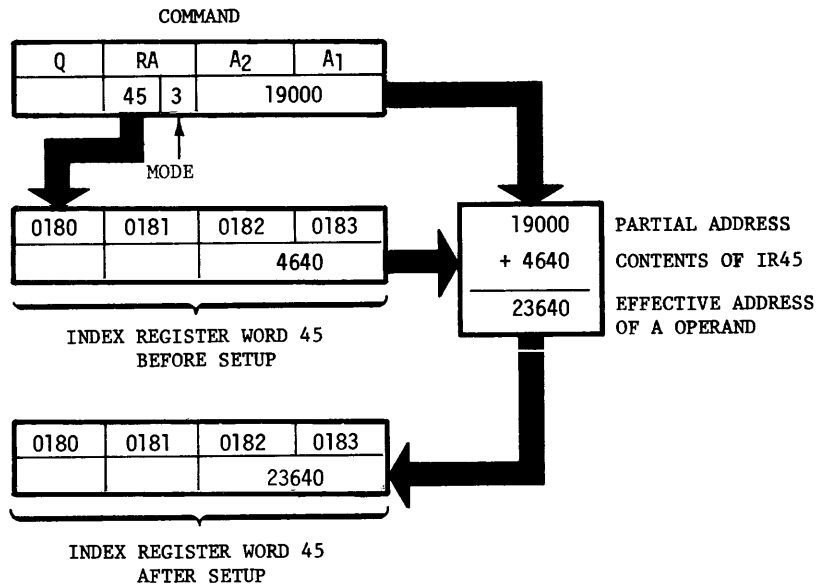
EXAMPLE 2:

Mode 3   No indexing

COMMAND

10000	10001	10002	10003
Q	RA	A₂	A₁
	0	3	17491

↑ MODE

NO INDEX
REGISTER
SPECIFIED

EFFECTIVE ADDRESS OF A OPERAND
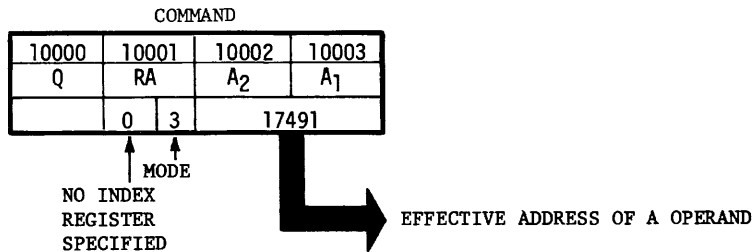
A2A1 Characters

This is a 2-character binary field representing the partial address of the A operand. If the RA character designates neither indexing nor indirect addressing the A2A1 characters form the effective address of the A operand.

Length - T

The binary value of the T portion of the command specifies the field length, in bytes, of the A and B operands. T is an 8-bit character ranging in value from 0 to 255 with 0 usually considered equal to 256. There are some exceptions that will be discussed in detail for each command in the NCR Century 200 command section.

Index Register - RB

The RB character is identical to RA, except that it pertains to the B operand rather than the A operand.

B2B1 Characters

The B2B1 characters are identical to A2A1, except that they pertain to the B operand rather than the A operand.

NOTE

> Any mode of addressing, with or without indexing, may be used to derive the effective A operand address and effective B operand address, independent of each other.

Command Example (Values in hexidecimal notation)

Q	RA	A2	A1	T	RB	B2	B1
64	58	02	BC	05	70	00	65

Partial B
address = 00101

IR28
Mode 0

Field Length = 5 bytes

Partial A
address = 00700

IR22
Mode 0

Command code for MOVE A RIGHT TO LEFT command (2-address)
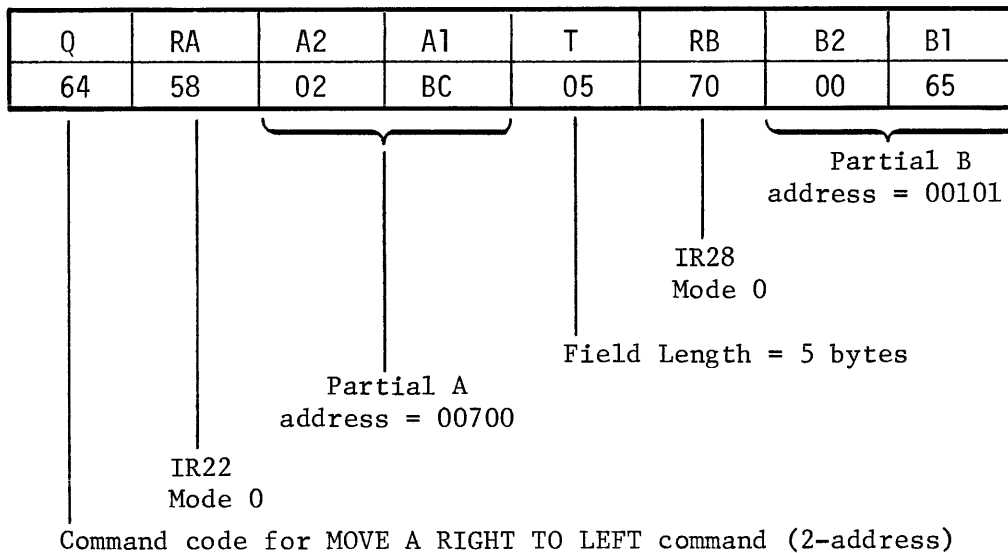
Assume that the two index registers contain the following information:

(IR22) = OAFO (02800)
(IR28) = 0898 (02200)

OAFO + 02BC = ODAC (03500) Effective A address
0898 + 0065 = 08FD (02301) Effective B address

Operand contents before command execution:

	03500	03501	03502	03503	03504
A =	00110100	00110001	00110010	00110110	00111001

	02301	02302	02303	02304	02305
B =	00110001	00110010	00110110	00111001	00110000

Following execution of the MOVE A RIGHT TO LEFT command, the initial contents of A have replaced the initial contents of B. The contents of A are unchanged:

	03500	03501	03502	03503	03504
A =	00110100	00110001	00110010	00110110	00111001

	02301	02302	02303	02304	02305
B =	00110100	00110001	00110010	00110110	00111001

Since the command specified no incremental indexing, the contents of IR22 and IR28 remain the same. If the command had specified incremental indexing for both operands, then the contents of IR22 would be ODAC, and the contents of IR28 would be 08FD at the termination of the command.

Implied T and B Operation

All commands that terminate normally have predictable T and B values available for use by subsequent commands. Except for the RESTORE command, the T value available after command execution is the one available following command setup. The B value is more variable and depends upon the specific command executed; it too, however, is predictable according to the conventions described in the NCR Century 200 command publication. Any command may be coded in a 1-address format with the second operand and the length implied, i.e., derived from the B and T characters established for a previous 2-address command. The setup of the 1-address command does not disturb these values, and they are used as if the current command had set them up. This characteristic permits strings of 1-address commands to be "chained" to a 2-address command as illustrated in the following example.

EXAMPLE:

Implied T and B Operation

Commands are in consecutive memory locations.

1.

Q	RA	A2	A1	T	RB	B2	B1
64	00	03	E8	02	00	07	D0

MOVE A RIGHT TO LEFT - Each character in the field specified by the A address is moved into the field specified by the B address, one character at a time, starting with the rightmost character.

2.

Q	RA	A2	A1
C0	00	0B	B8

ADD SIGNED - The A field and the B field are added decimally, one character at a time, and the result replaces the B field. Each field contains signed, packed, decimal information.

3.

Q	RA	A2	A1
C5	00	0F	A0

COMPARE SIGNED - The A field is compared to the B field. Both fields are considered to contain signed, packed information.
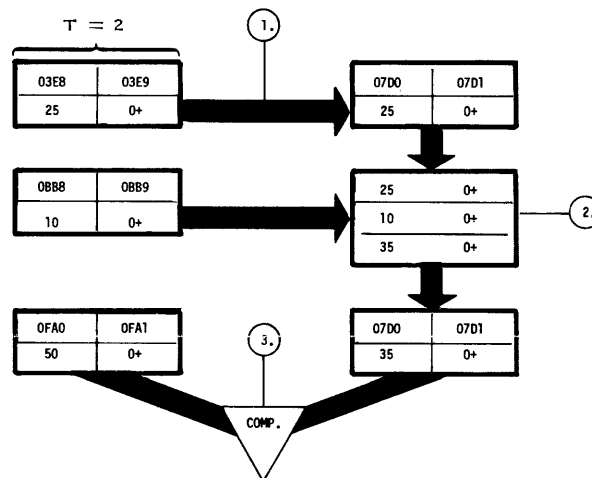
ASYNCHRONOUS OPERATION

Internal speed of the NCR Century 200 is about five times that of the NCR Century 100. This speed results from asynchronous operation; that is, an operation starts as a result of a signal that the previous operation has been completed, without waiting for clocked intervals when memory access is not desired. Because of this, the processor is able to transfer information among its registers and to test various flags and indicators without a clocking scheme. These registers are called live registers and their contents are available to the processor without accessing memory.

Live Registers

The information contained in the live registers is originally obtained from memory. Once this information has been transferred to a live register, the processor is able to access and to operate on it with no time delay. When the manipulation of the data is complete, a memory cycle transfers new information to the live registers. In effect, the contents of live registers can be accessed in almost zero time. These registers are listed in the following illustration.

Memory Registers

Memory registers, stored in reserved memory areas of internal memory, contain program status words, error status word, control words, and reserved software information. The 63 standard index registers are included in this area. The processor uses one memory cycle to obtain two bytes of information from these registers, as well as to write two bytes of information into these registers.

Special Registers

Special registers are added to the processor hardware when particular options are installed. These registers perform specialized functions related directly to the option. The base address and limiting address registers (BAR/LAR) required by multiprogramming are examples of the special registers needed.

PROCESSOR LIVE REGISTERS

TERM	NAME	STD OR OPT	CONTENTS
CR	SEQUENCE CONTROL REGISTER	STD	Address of the next command to be executed. The CR is incremented during the command setup phase.
LA	EFFECTIVE A ADDRESS REG.	STD	Address location of the A operand.
LB	EFFECTIVE B ADDRESS REG.	STD	Address location of the B operand.
LC	MISCELLANEOUS REGISTER	STD	Used during command execution when an address must be stored temporarily.
L	MEMORY ADDRESS REGISTER	STD	Address of the memory location to be accessed.
F	F REGISTER	STD	A 16-bit input to the adder.
G	G REGISTER	STD	A 16-bit input to the adder.
J	J REGISTER	STD	The 16-bit adder output.
MA	MEMORY OUTPUT REGISTER	STD	Contains one byte output from memory.
MB	MEMORY OUTPUT REGISTER	STD	Contains one byte output from memory.
MC	MEMORY INPUT REGISTER	STD	Contains one byte input to memory.
MD	MEMORY INPUT REGISTER	STD	Contains one byte input to memory.
TT	T REGISTER	STD	Original T of command; changed during command setup only.
T	TALLY REGISTER	STD	T character; counted up or down during command execution.
Q	Q REGISTER	STD	Q of the command. It is changed during command setup.
BAR	BASE ADDRESS REGISTER	OPT	Starting address of a memory location used in a multiprogramming option.
LAR	LIMIT ADDRESS REGISTER	OPT	Limiting address for a multiprogramming option. Prevents access by accident to a different program.
MN	MONITOR REGISTER	OPT	Address of a memory location. A trap condition occurs when this location is written into.

FLAGS

Stored in live registers in the processor is a set of standard hardware flags
that can be accessed by the processor with no effective time delay. These
flags are used by the processor to denote the condition or result of an oper-
ation. Flag status is monitored and used by the processor for making logical
decisions concerning a given operation. A flag is either ON or OFF, with ON
meaning a certain condition or result did occur. Standard flags included in
the NCR Century 200 Processor are the comparison set of greater than, less
than, and equal to, plus the overflow flag which signals that an arithmetic
operation has resulted in a quantity greater than the specified storage area.

The standard and special-purpose flags which are added to the processor when
required by optional features, are listed in the following illustration.

PROCESSOR FLAGS			
TERM	NAME	STD OR OPT	FUNCTION
L	LESS THAN	STD	Indicates a less than condition during a compare or scan command.
E	EQUAL TO	STD	Indicates an equal to condition during a compare, scan or a decode-to-delimiter command.
G	GREATER THAN	STD	Indicates a greater than condition during a compare command, or a decode to delimiter command.
OF	OVERFLOW	STD	Indicates a condition where the adder function resulted in an output bit condition greater than the output register (J) of the adder.
S	USER SUPERVISOR	OPT	Included in the multiprogramming option. Indicates the state of the processor.
A	FLAG A	OPT	Used in conjunction with the S flag. If flag A is on it indicates BAR/LAR processing for the A value.
B	FLAG B	OPT	Used in conjunction with the S flag. If flag B is on it indicates BAR/LAR processing for the B value.

INDICATORS

Indicators, like flags, provide the processor with information denoting the condition or result of an operation. The following table names both the indicator and its function.

PROCESSOR INDICATORS			
TERM	NAME	CONSOLE INDICATOR	FUNCTION
RI	REPEAT INDICATOR	YES	Indicates a repeat of a command.
IP	INTERRUPT PERMIT	YES	Set on by the program to permit interrupt of the main program to handle peripheral termination.
II	INTERRUPT INDICATOR	YES	Indicates that a peripheral termination occurred. If IP was not set on previous to it, the interrupt will not be serviced.
TP	TRACE PERMIT	YES	Indicates that each command executed is being monitored.
ME	MEMORY ERROR	YES	Indicates that an error was encountered while reading data from memory.
PE	PROGRAM ERROR	YES	Indicates an error condition other than a memory error.
EI	ERROR INDICATOR	YES	Indicates an unrecoverable error condition. If EI and either ME or PE is on the processor halts.
CCI	COMMAND CODE INDICATOR	NO	Indicates an unrecognized command code and a trap routine is entered.

FUNCTIONAL OPERATION

Control

The ALU operates under the guidance of a control section. When the command is decoded, the control section is activated to regulate data transfer among the data storage registers, to and from memory, through the adder, and to the addressing logic.

The control section is divided into three areas: N, P, and miscellaneous decision logic. The N control logic regulates the processor as it proceeds through the series of steps required to execute the command. These steps are called N flows and may be separated into two groups: the flows involved with setting up and interpreting the command, and those required to execute the command.

Within each N flow there is a series of computer cycles called P counts. The number of P counts within an N flow varies from 1 to 10, depending on the function to be performed. An individual P count performs a logical step such as memory access or data transfer between registers. The duration of a P count is variable: if memory is accessed, the duration of the P count equals the duration of the memory cycle; however, if memory is not accessed (for instance, data is transferred between live registers), the duration of the P count is less, restricted only by the logic circuit delays.

The miscellaneous decision logic controls the P counter and other logic pertinent to the particular command, as it guides the processor through the necessary N flows to accomplish the desired function.

Addressing

As the processor proceeds through the various N flows in the setup and execu-
tion of a command it is necessary to access memory locations to obtain the
pertinent data to complete the command. The addressing function is accomplish-
ed by the L register. As indicated by the unshaded data paths in the following
illustration, the L register receives its information from the CR, LA, LB or
LC registers depending upon the function being performed. The CR register con-
tains the address of the next command to be executed, the LA register contains
the address of the A operand, and the LB register contains the address of the
B operand. The LC register is generally used for temporary storage of an ad-
dress.

ADDRESS FLOW TO THE ADDRESS [L] REGISTER

Adder

The center of the processor hardware operations is an area called the adder. The primary function of the adder is arithmetic operations. Within this area the processor performs addition, subtraction, and comparison.

The unshaded data paths in the following illustration indicate the adder data flow. For example, in performing addition, the address of the first A field character is calculated, and a read memory cycle puts the contents of this memory location in either the MA or MB data register. This character is transferred through the adder, via the adder input (G) and output (J) registers to the LC register. Then the address of the first B field character is calculated, and a read memory cycle puts this character in either the MA or MB data register. The addition is performed by putting the contents of the LC into F, and the contents of MA or MB into G. The result is output through the J register and transferred to the MC register. A write memory cycle places this data into the B operand memory location. This process is repeated until the specified number of A and B field characters are added.

DATA FLOW THROUGH THE ADDER
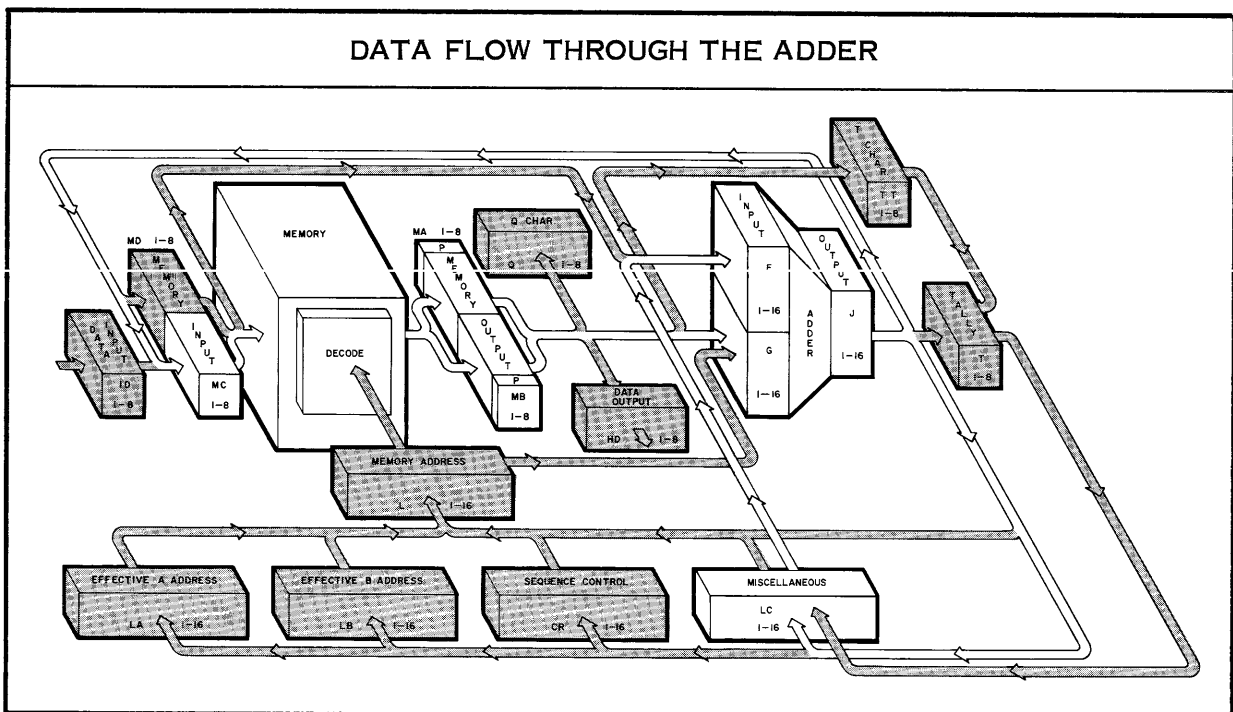
Data Registers

The unshaded data paths in the following illustration indicate the flow of
data to the various data registers.

The MA and MB registers are temporary storage devices for information output
from memory. This information is then distributed to the appropriate register
depending on the function to be performed.

The Q register receives the Q character of a command from the MB data register
and holds it until needed.

The TT register receives the T character (length) of a command and when re-
quired passes it along to the T register.

The T register has various uses, such as temporary storage of a character to
be compared to another or incrementing or decrementing the length character
(T). The MC and MD registers are used to input data to memory during a write
cycle.

The ID register receives data from the peripherals to be input to memory; the
HD register outputs data from memory to peripherals via the MA and MB registers.

DATA FLOW TO THE DATA REGISTERS
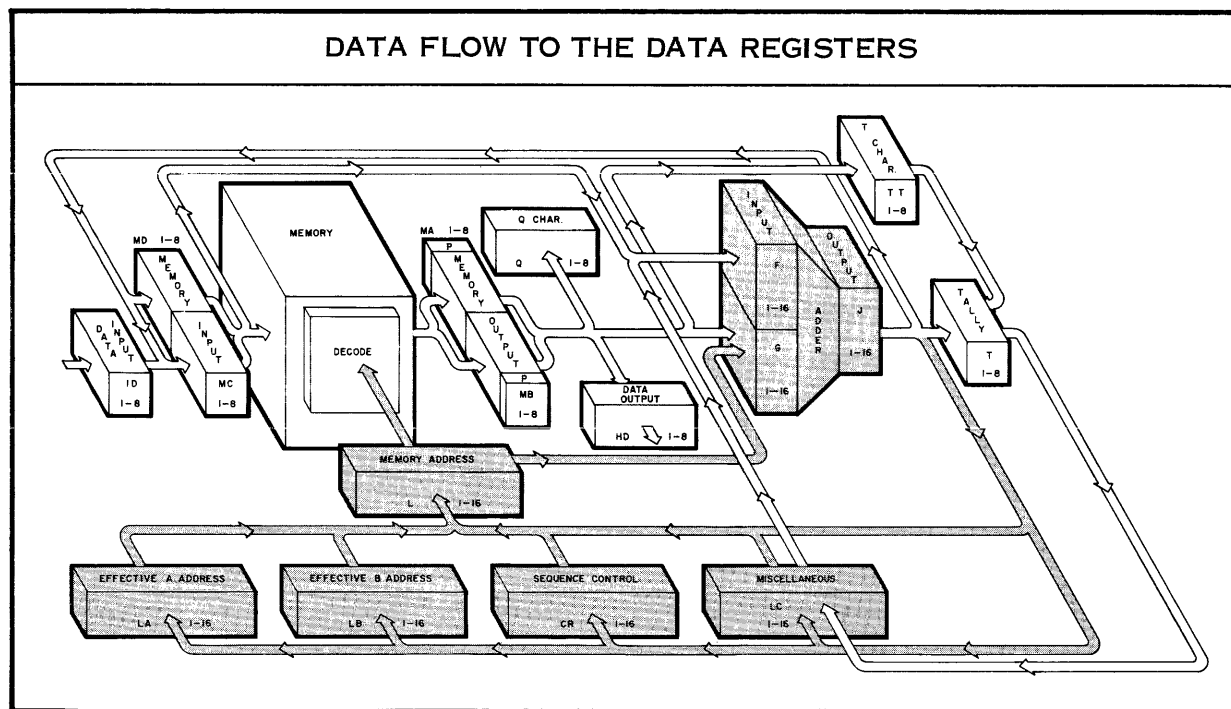
Expanded Registers

The illustration below shows the processor registers as they appear in the base system.

With the extended memory option these registers are expanded as follows:

BASIC	EXPANDED
L 1-16	L 1-19
LA 1-16	LA 1-19
LB 1-16	LB 1-19
LC 1-16	LC 1-19
CR 1-16	CR 1-19
MB 1-8	MB 1-8, MB 10-12
MD 1-8	MD 1-8, MD 10-12
F 1-16	F 1-19
G 1-16	G 1-19
J 1-16	J 1-19

BASE SYSTEM REGISTERS
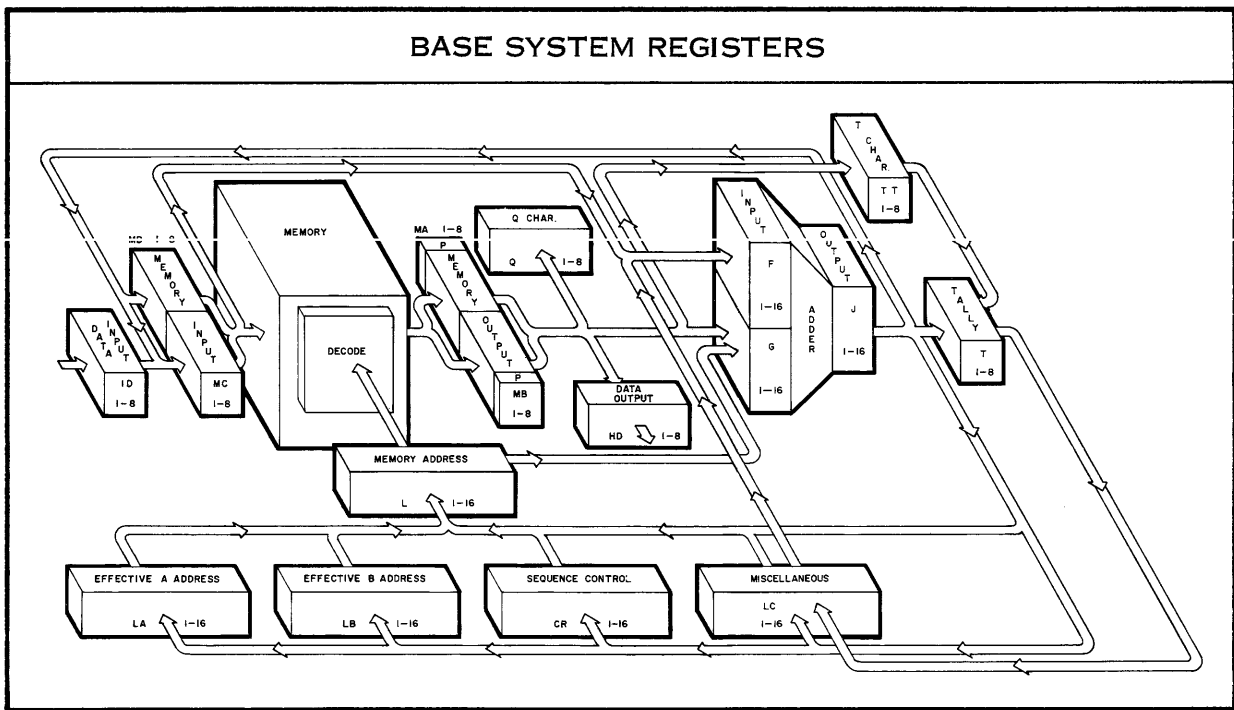
Command Setup and Execution

In performing the function specified by the command, the NCR Century 200 processor logic proceeds through three major phases: command setup, command execution and between commands testing (BCT). The BCT procedure is explained in detail under "Between Commands Testing," in this publication.

During the command setup phase, which includes command interpretation and indexing (if specified), the command address is transferred from the sequence control register (CR) to the memory address register (L) and tested for size and validity. If the address is greater than memory size or not evenly divisible by 4 (0 modulo 4), the processor enters a program error routine.

If no error is detected, the processor then reads the Q and RA characters of the command and distributes them to the appropriate registers. Bits 1 and 2 of the RA character are tested and set the corresponding hardware flags for the mode of addressing. The Q character is examined for legality. If it is not recognized as a part of the command set, it is considered invalid and the command code indicator (CCI) subsequently forces the processor to enter a trap routine to handle the command. The trap routine (a software function) examines the command to see if it is an interpretive. An interpretive command is one that is not hardware recognized but which may be performed by software (such as multiply, if the multiply feature is not incorporated in the system). If it is not an interpretive, a software PE routine is entered.

The A2 and A1 characters are read from memory and stored in the LA register (address of the A operand). If b8 of the Q character specifies a single address format and no indexing is required, the processor enters the command execution phase. This is possible because the previous two-address command has had the T, RB, and B2B1 characters distributed to the appropriate registers, making them available for immediate use by the command being performed.

If indexing is specified, the mode is examined and the proper N flow executed. See "Index Register - RA", in this publication, for an explanation of the various modes of addressing.

If the command has a two-address format, the T, RB, and B2B1 characters must be set up. The T character is stored in the TT register. The processor handles the RB and B2B1 characters for the B operand address in the same manner as RA and A2A1 characters.

The entry into the individual N flows for execution is dependent upon the interpretation of the Q character. The actual steps involved in executing a given command are subject to too many variables to permit a detailed explanation in this publication; refer to the supplementary publication on commands for additional information. Once any command has been executed the processor enter the BCT flow. If the program flow is not changed in the BCT flow, the processor accesses the next command in sequence and enters the setup phase.

BETWEEN COMMANDS TESTING

Introduction

Between commands testing (BCT) is a term given to a series of tests made by the processor upon termination of a command. This testing gives the processor the ability to alter its program flow, halt, or continue the same program flow, depending upon the results of the tests.

Functional Operation

If, during between commands testing the conditions tested warrant it, the processor can alter its program flow by entering a trapping flow. The particular trapping flow entered is dependent upon the results of the between commands testing. While in a trapping flow the contents of the control register (CR) are changed to the address of a software trapping routine designed to process the particular condition discovered during the between commands testing. The trapping routine addresses are held in reserved memory control areas. (See "Memory," in this publication.)

The significant state of the processor at the time of trapping is stored in reserved index registers, as the status word. Storage of the status word enables the processor to later re-enter the program flow at the point where the trap condition occurred. There are two types of status words, program status words (PSW) and error status words (ESW). (See "Memory," in this publication.)

The one exception to the functional operation just described is the Repeat Flow, which is to be explained in detail later in this section.

The conditions that are tested during between commands testing, and the sequence in which they are tested (priority) are as follows:

1. Error Indicator (EI) and one of the following

   ● Memory Error (ME)
   ● Programming Error (PE)
   ● Trapping Command Code - Command Code Indicator (CCI)

2. Memory Error (ME)
3. Programming Error (PE) and/or Trapping Command Code - Command Code Indicator (CCI)
4. Repeat Indicator (RI)
5. Trace Permit (TP)
6. Interrupt Indicator (II) and Interrupt Permit (IP)
7. Halt

If any of the above conditions are met (ON) the processor will either enter a trapping flow to alter its program flow or halt. If none of the above conditions are met, the processor will continue the normal program flow.

Error Indicator (EI)

The error indicator (EI) causes a processor error halt if a command malfunction occurs at a time when the program cannot process it.

Command malfunctions are those malfunctions detected during command setup, execution, or between commands, rather than those detected by the I/O or printer control. These command malfunctions include memory errors (ME's), program errors (PE's) and trapping command codes - command code indicators (CCI's).

The error indicator (EI) is set ON in the ME trapping flow and in the PE and CCI trapping flow if a PE has been detected. EI is not set ON for a CCI. The error indicator is set OFF by the JUMP command and by the console RESET switch.

If, during between commands testing, the error indicator (EI) is ON, and ME, PE, or CCI is ON the processor enters the error halt state.

Once the processor enters the error halt state, the I/O control becomes inactive; peripherals terminate their activity as they would if their request for service was not answered.

Memory Error (ME)

The detection of a memory error causes the ME Indicator to be set ON. The erroneous character remains undisturbed, i.e., it still contains an incorrect parity, except in two instances:

- The character containing the ME is one of the index register characters used when incremental indexing (mode 3) has been specified.
- The character containing the ME is one of those characters which contain information in the error status word.

In both instances, the character containing the ME will be replaced by a different character with correct parity before the next command is accessed.

If during between commands testing the error indicator (EI) is OFF, and the memory error (ME) is ON, the processor enters the ME trapping flow. Upon entering this trapping flow, EI is set ON and ME is set OFF. PE is set OFF, if it is ON.

The significant state of the processor, error status word (ESW), is stored in memory locations 020-031; IR 5, 6, and 7.

The repeat indicator (RI) and overflow flag (OF) are set OFF. The interrupt permit indicator (IP) is set OFF. For systems with the trace option, the trace permit (TP) is set OFF. For systems with the multiprogramming option the S Flag is set ON.

The control register (CR) is loaded from memory locations 257-259. These locations contain the ME trap address, the address of the first command of the ME trap routine. Command setup is then entered for this command.

ME ON

EI SET ON
ME SET OFF
PE SET OFF

ESW ——>IR5, 6, & 7

RI & OF SET OFF
IP SET OFF
TP SET OFF *
S FLAG SET ON **

(257-259)   CR
ME TRAP ADDRESS

GO TO
COMMAND
SETUP
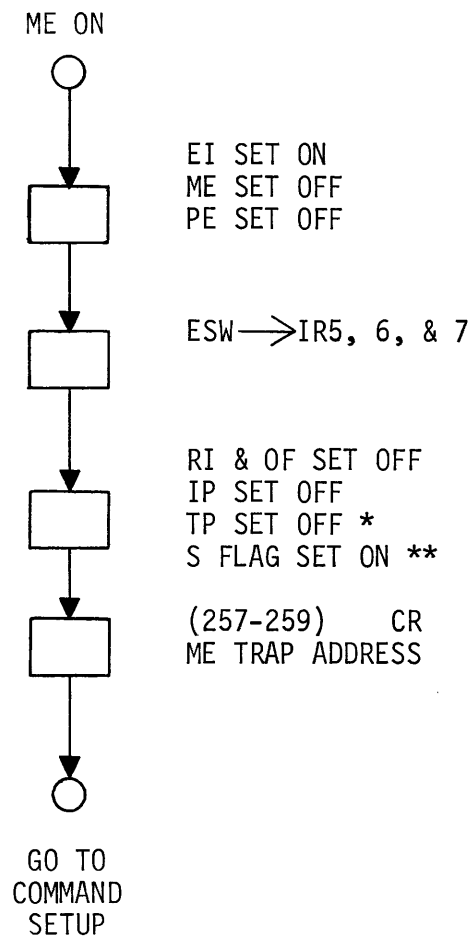
*   In systems with the
     Trace Option

**   In systems with the
     Multiprogramming
     Option

Program Error (PE) and/or Trapping Command Code - Command Code Indicator (CCI)

The detection of a programming error during command setup and execution causes the PE indicator to be set ON. The current operation is terminated and between commands testing is begun.

A trapping command code is a command code which cannot be executed by hardware alone. The detection is made by the ALU. A command cannot be executed by hardware alone if it is not included in the set of hardware commands available for a particular system or if it is privileged (explained in detail under the multiprogramming option). The detection of a trapping command code causes the command code indicator (CCI) to be set ON, the current operation terminated, and between commands testing begun.

If, during between commands testing, the program error (PE) and/or the command code indicator (CCI) is ON, the PE and command code trapping flow is entered. Depending upon the cause of the trap, one of the following error codes will be stored in memory location 036:

Type of Error	Memory Location 036
	b4 b3 b2 b1
PE	0 0 0 1 = 1
Trapping Command Code (not privileged)	0 0 1 0 = 2
PE and Trapping Command Code (not privileged)	0 0 1 1 = 3
PE caused by characteristic overflow in systems with the Floating Point Option.	0 1 0 1 = 5
Trapping Command Code caused by privileged command occuring in user state – Multiprogramming Option	1 0 0 0 = 8
PE and Trapping Command Code caused by privileged command occuring in the user state	1 0 0 1 = 9

If the trap was caused by a PE, the PE indicator is set OFF, the error indicator (EI) is set ON, and the error code is stored in memory location 036.

If the trap was caused by a trapping command code, CCI is set OFF, EI is left undisturbed, and the error code is stored in memory location 036.

If the trap was caused by both a trapping command code and a PE, the CCI and PE are set OFF, EI is set ON, and the error code is stored in memory location 036.

The significant state of the processor, error status word (ESW), is stored in memory locations 020-031; IR 5, 6, and 7.

The repeat indicator (RI) and overflow flag (OF) are set OFF. The interrupt permit indicator (IP) is set OFF. For systems with the trace option, the trace permit (TP) indicator is set OFF. For systems with the multiprogramming option, the S Flag is set ON.

The control register (CR) is loaded from memory location 261-263. These locations contain the PE and command code trap address, the address of the first command of the PE and command code trap routine. Command setup is then entered for this command.

PE AND/OR CCI

PE AND CCI SET OFF
EI SET ON (WITH PE ONLY) ERROR CODE 036

ESW $\longrightarrow$ IR5, 6, & 7

RI & OF SET OFF
IP SET OFF
TP SET OFF *
S FLAG SET ON **

(261-263) $\longrightarrow$ CR
PE & COMMAND CODE TRAP ADDRESS

GO TO
COMMAND
SETUP
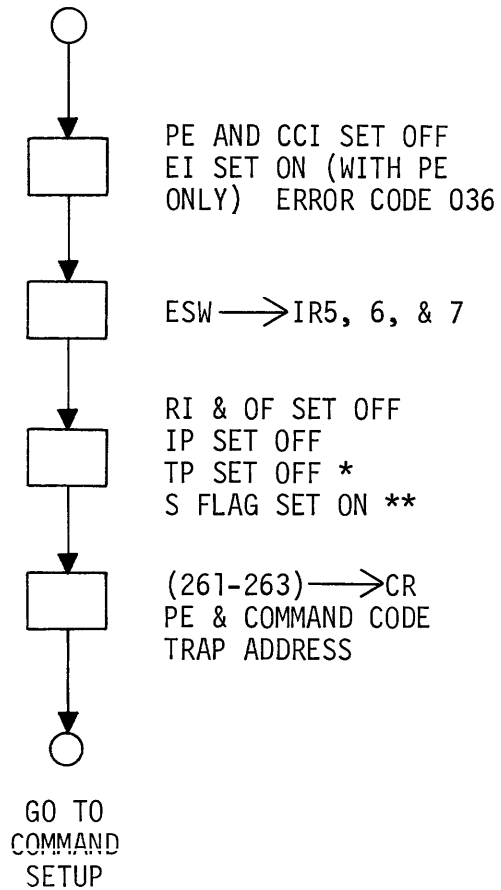
*   In systems with the Trace Option

**   In systems with the Multiprogramming Option

Repeat Indicator (RI)

The repeat indicator (RI) is set ON during the REPEAT command if the number of times specified for execution of the next command in sequence (command to be repeated) is greater than 0.

A repeat counter (RC) stores the number of times a command is to be repeated. This counter is decremented by 1 each time the command being repeated is executed. When the repeat counter (RC) equals 0, repeating terminates.

A secondary repeat indicator (RII) is also set ON during the REPEAT command. It is used when one of the following commands is to be repeated:

    BINARY COMPARE
    COMPARE SIGNED
    TEST BIT
    TEST CHARACTER EQUAL
    TEST CHARACTER UNEQUAL
    SCAN
    TABLE COMPARE

If the conditions as stated in one of these commands are satisfied before the repeat counter (RC) becomes equal to 0, the secondary repeat indicator (RII) is turned OFF, allowing the repeating to terminate.

If the repeat indicator (RI) is ON following the execution of any command except the REPEAT and RESTORE commands, the between commands repeat flow is entered.

The repeat counter (RC) memory location 032, is read out and tested for 0. If it is 0, an immediate PE results.

If the repeat counter (RC) is greater than 0, it is decremented by 1 and re-stored to memory .

The repeat counter (RC) is then again tested for 0 and the secondary repeat indicator (RII) is tested.

If the test indicates that the repeat counter (RC) is 0 or that the secondary repeat indicator (RII) is OFF, the repeat indicator (RI) is set OFF and the operation terminates. The control register has been updated to address the next command; command setup is then entered for this next command.

If the test indicates that the repeat counter (RC) is not 0 and the secondary repeat indicator (RII) is ON, the control register (CR) is decremented by the size of the command just executed, thereby permitting the same command to be re-executed. Command setup is then entered for this same command.

RI ON

IS RC = 0?

PE CONDITION

DECREMENT RC BY 1

IS RC = 0 OR RII OFF?

RI SET OFF

CR DECREMENTED BY THE SIZE OF COMMAND JUST EXECUTED

GO TO
COMMAND
SETUP
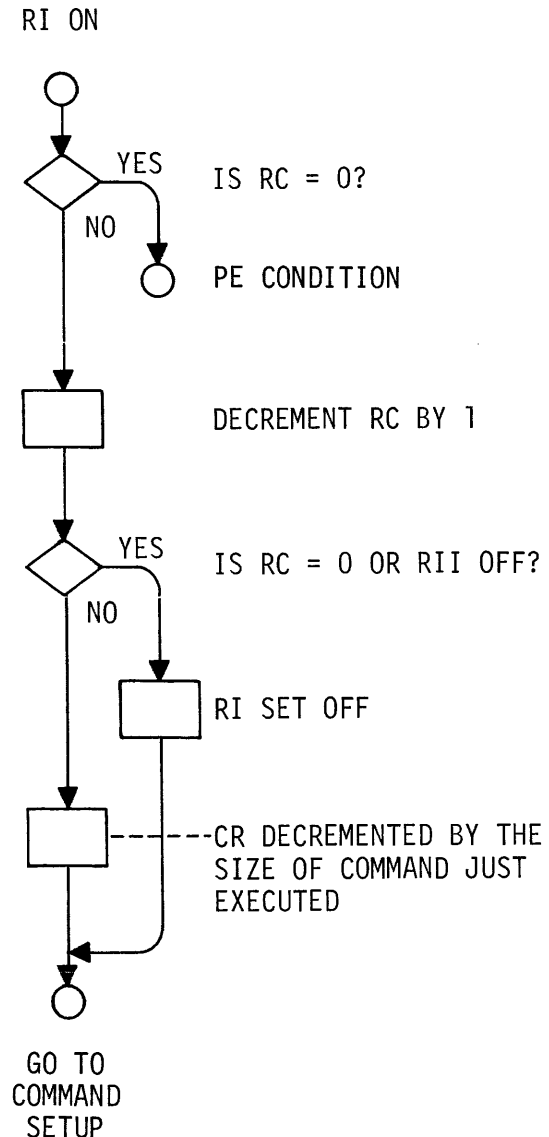
Trace Permit (TP)

The trace option provides the ability to monitor each command execution when the trace permit (TP) flag is ON. The option includes three additional machine commands, two of which can be used for setting or resetting of the trace permit (TP) flag. The third command gives the trace option the ability to specify a memory location that is monitored when the MONITOR switch is ON. Writing into this memory location causes trapping to occur by setting the trace permit (TP) flag ON. This memory location may also be specified from the console.

The trace permit (TP) flag is tested by the processor during between commands testing, and, if it is ON, the trace interrupt flow is entered.

Upon entering the trace interrupt flow, the significant state of the processor, program status word (PSW), is stored in memory locations 052-063 (IR13, 14, and 15).

The overflow flag (OF), repeat indicator (RI) and trace permit (TP) are set OFF. If the system has the multiprogramming option the S flag is set ON.

The control register (CR) is loaded with the contents of memory locations 277-279, the address of the first command of the trace trap routine. Command setup is then entered for this command.

TP ON

PSW → IR13, 14, & 15

OF, RI & TP SET OFF
S FLAG SET ON *

(277-279) → CR
TRACE TRAP ADDRESS

GO TO
COMMAND
SETUP
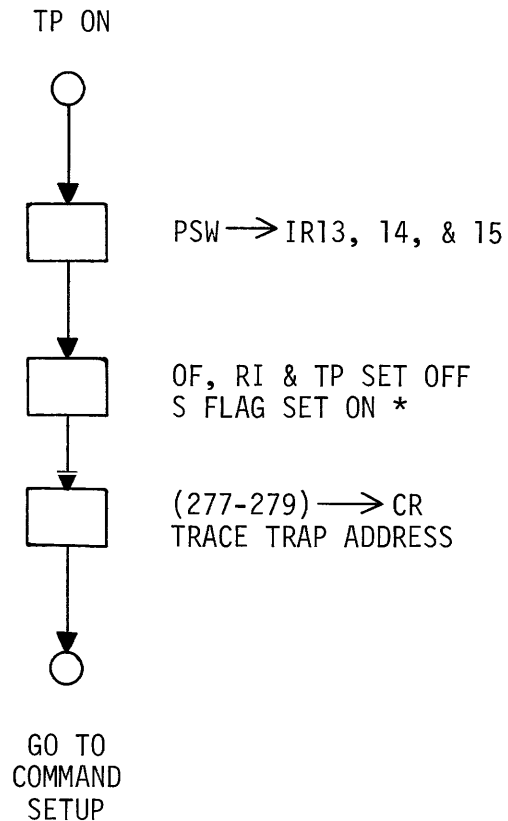
*   In systems with the
    Multiprogramming Option

Pressing the LOAD button causes the trace permit (TP) to be turned OFF.

Between commands testing provides the ability to interrupt the normal flow of the program to enable the interrupt routine to process the termination of I/O operations.

Interrupt permit (IP) may be set ON or OFF by program control to indicate the program's readiness to accept an interrupt. IP is set ON by the system software at the proper time. IP is set OFF by the SET IP OFF command, by any trapping flow (interrupt, ME, PE, or command code), or by the console LOAD button.

The interrupt indicator (II) is used to remember that an interrupt situation has occurred, until such time as the interrupt can be serviced. An interrupt situation occurs whenever the I/O control or printer control detects a terminating status signal or when a latent error condition occurs (see I/O Control section).

The processor enters the interrupt trapping flow during BCT if IP and II are both on.

IP AND II ON

Upon entering the trapping flow, IP and II are both set OFF.

The significant state of the processor, program status word (PSW), is stored in memory location 040-051 (IR10, 11 and 12).

IP & II SET OFF

PSW ——>IR10, 11 & 12

The overflow flag (OF) and repeat indicator (RI) are set OFF. The trace permit (TP) is set OFF in systems having the trace option. The S flag is set ON in systems having the multiprogramming option.

OF & RI SET OFF
TP SET OFF *
S FLAG SET ON **

(273-275) ——>CR
INTERRUPT TRAP ADDRESS

The control register (CR) is loaded with the contents of memory locations 273-275, the address of the first command of the interrupt trap routine. Command setup is then entered for this command.

GO TO
COMMAND
SETUP
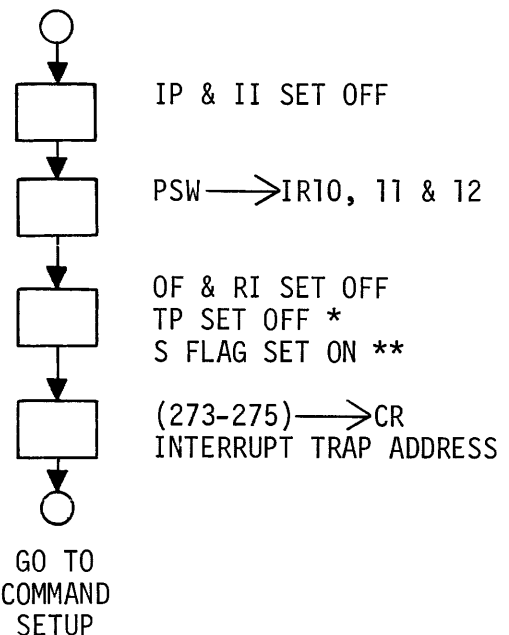
*   In systems with the
    Trace Option

** In systems with the
    Multiprogramming
    Option

Halt

If the HALT button is ON upon completion of a command (during between commands testing), the processor enters the halt state. Upon exit from the halt state, between commands testing again takes place.

I/O CONTROL

INTRODUCTION

NCR Century 200 I/O operation is initiated by the ALU, using information de-
rived from the INOUT command. After the ALU initiates peripheral selection on
one of the system's trunks, the proper response from the peripheral unit frees
the ALU to issue another INOUT command for a peripheral on another trunk or
to return to processing subsequent program instructions. In either case, the
I/O control takes charge of all processor/peripheral communication. An I/O
termination must occur on one of the selected trunks before subsequent INOUT
commands can be processed on that trunk.

Data is input and output serially, by byte. The parity bit is checked by
the I/O control when data is received from the peripheral and by the periph-
eral unit itself when it receives data from the I/O control.

If the I/O control and the ALU simultaneously request a memory cycle, the
I/O control takes priority. Since peripheral servicing does not require all
the memory cycles available within a given time frame, the ALU "steals"
unused cycles and continues processing during data transfer. This character-
istic provides the NCR Century 200 with effective 5-way simultaneity, which
allows four I/O operations and an ALU operation to take place at the same time.

GENERAL DESCRIPTION

Trunks

- #### Common Trunk Concept

    The input/output trunk which is called the common trunk, provides pro-
    cessor control lines that are logically separated from data lines. The
    term common trunk does not indicate a single trunk but refers to the
    concept that the processor communicates with all peripherals in the same
    way. Actually, there can be up to eight separate trunks in this common
    trunk concept. In order for the processor to communicate with all
    peripherals in the same manner, the common trunk interfaces are located
    in the peripheral units.

    The I/O control, working in conjunction with the common trunk, provides
    4-way I/O simultaneity for the quadraplex system and 8-way I/O simultaneity
    for the octaplex system.

- #### Trunk Assignments

    Trunks 0 and 7 are reserved for integrated peripherals. An integrated
    peripheral shares logic and power supplies with the processor.

    - Trunk 7 is always reserved for the integrated printer.
    - Trunk 0, position 0 is reserved for the card or tape unit (COT).
    - Trunk 0, position 1 is reserved for the console input switches.

- Trunk 0, position 5 is reserved for the I/O writer. The remainder of trunk 0 is not available for other peripherals.
- All other trunks are available for any freestanding peripherals that do not exceed the bandwidth for the specific trunk.

Trunks are serviced according to an I/O priority scheme; the higher the trunk number, the higher the trunk priority. There is one exception: to avoid system overload, the COT on trunk 0 has second highest priority. All other units on trunk 0 have lowest priority. When two trunks request service at the same time, priority is assigned in the following order:

- Trunk 7
- COT on Trunk 0, Position 0
- Trunk 6
- Trunk 5
- Trunk 4
- Trunk 3
- Trunk 2
- Trunk 1
- Trunk 0, excluding COT

- High-Speed Trunks

    As an option, either trunk 6 or both trunks 5 and 6 may be converted to high-speed trunks. Conversion to high-speed trunks makes it possible to use high-speed peripherals on those trunks. By definition, a high-speed peripheral is one whose data transfer rate exceeds 120KB.

System Configurations

- Base System (Quadraplex)

    The base system, which consists of four input/output trunks, is called a quadraplex system. Each of the four trunks (0, 5, 6, and 7) provides processor control lines and data lines to the peripheral. The trunk carries the data and control information both ways, from the processor to the peripheral and from the peripheral to the processor. Trunk 0 and trunk 7 are reserved for integrated peripherals; only trunk 5 and 6 are available for freestanding peripherals. In the quadraplex system, trunk 6, or trunks 5 and 6 together, may be used as high-speed trunks; trunk 5 may not be used as a high-speed trunk by itself.

QUADRAPLEX SYSTEM

```
┌──────────┬──────────┐   Trunk 0      [PO] [P1]                    [P5]
│          │          │                COT  Input              I/O Writer
│ MEMORY   │          │                     Switches
│          │   I/O    │   Trunk 5      [PO] [P1] [P2] [P3] [P4] [P5] [P6] [P7]
├ ─ ─ ─ ─ ─┤ CONTROL  │
│          │          │   Trunk 6      [PO] [P1] [P2] [P3] [P4] [P5] [P6] [P7]
│          │          │
│  ALU     │          │   Trunk 7      [Px]
│          │          │                Integrated
└──────────┴──────────┘                Printer
```
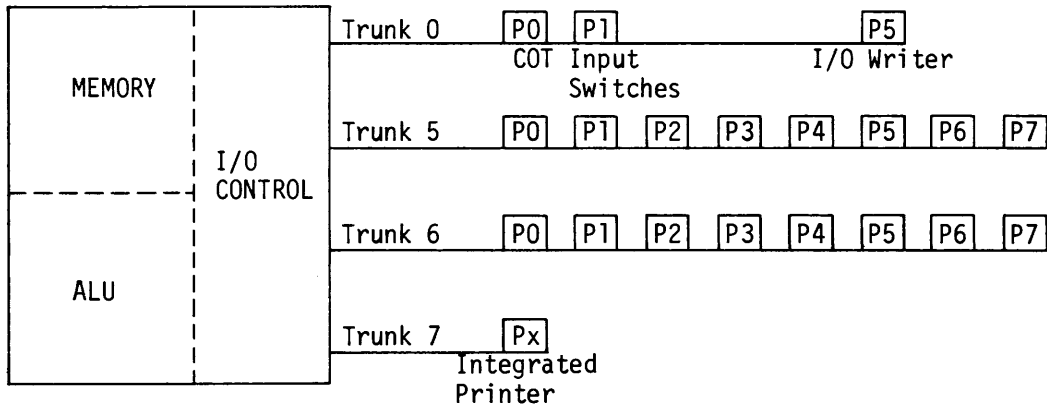
Trunk 0 is integrated with dedicated trunk positions for integrated peripherals.
Trunk 7 is integrated for printer only in any position.
Trunks 5 and 6 are for freestanding peripherals.

● Octaplex System

The system with eight trunks is called an octaplex system. The trunks are numbered 0 through 7. Trunks 0 and 7 are reserved for integrated peripherals. Trunks 6 through 1 are available for freestanding peripherals. In the octaplex system, trunks 5 and 6 are always high-speed trunks.

A quadraplex system can be upgraded to an octaplex system at the customer's site by the NCR technical representative.

OCTAPLEX SYSTEM

```
┌──────────┬──────────┐   Trunk 0      [PO] [P1]                    [P5]
│          │          │   Trunk 0 is integrated with dedicated trunk positions.
│          │          │   Trunk 1      [PO] [P1] [P2] [P3] [P4] [P5] [P6] [P7]
│ MEMORY   │          │
│          │          │   Trunk 2      [PO] [P1] [P2] [P3] [P4] [P5] [P6] [P7]
│          │          │
│          │   I/O    │   Trunk 3      [PO] [P1] [P2] [P3] [P4] [P5] [P6] [P7]
├ ─ ─ ─ ─ ─┤ CONTROL  │
│          │          │   Trunk 4      [PO] [P1] [P2] [P3] [P4] [P5] [P6] [P7]
│          │          │
│  ALU     │          │   Trunk 5      [PO] [P1] [P2] [P3] [P4] [P5] [P6] [P7]
│          │          │   Trunks 5 and 6 are high speed trunks.
│          │          │   Trunk 6      [PO] [P1] [P2] [P3] [P4] [P5] [P6] [P7]
│          │          │
│          │          │   Trunk 7      [Px]   Trunk 7 is integrated, for printer
└──────────┴──────────┘                       only located in any position.
```
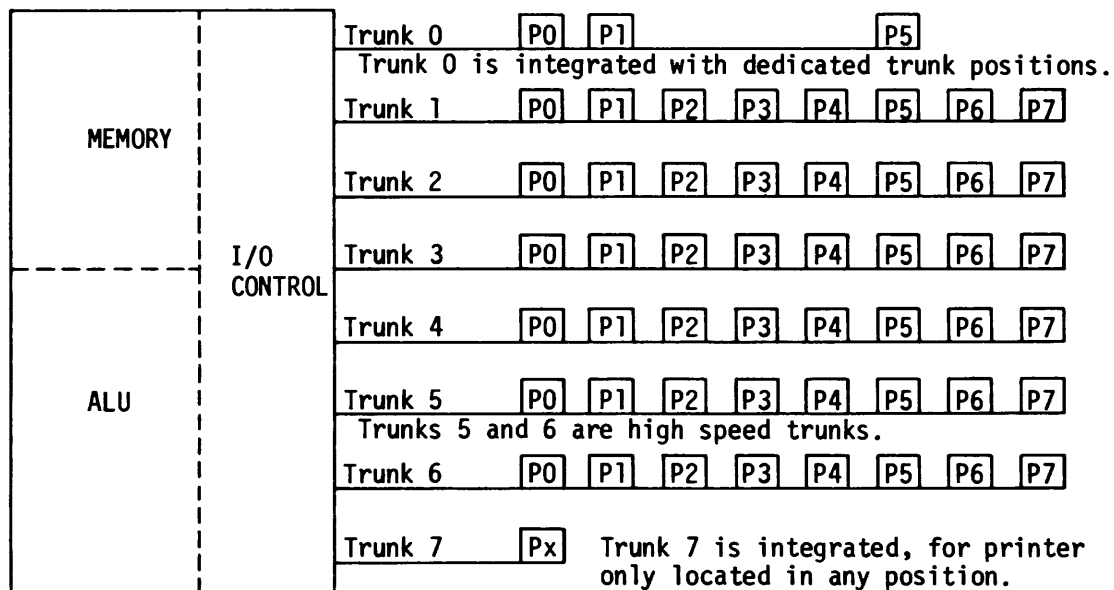
Trunks 1 thru 6 are for freestanding peripherals.

Data

A byte or character is made up of 8 data bits and a parity bit.

The byte is transferred on the data lines from the processor to the peripheral during an output operation or from the peripheral to the processor during an input operation.

Bandwidth

The combined activities performed by a computer system occur at varying speeds. Internal cycling, data transfer to and from memory, data transfer to and from peripherals, etc., all occur within the total available time. The term bandwidth is used to describe the range of speeds over which the various system components operate. The time limitations of the various functions are hardware-restricted, depending on memory cycling speed, internal processor speed, peripheral transfer speeds, and I/O handling speeds.

Generally, the maximum number of bytes that can be transferred in 1 second is called bandwidth.

● Trunk Bandwidth

Trunk bandwidth is generally defined as the maximum number of bytes that can be transferred on a given trunk. Memory cycling speed affects the trunk bandwidth; faster memories can handle data faster, consequently the trunk bandwidth is increased and vice versa.

The following table lists the various trunk bandwidths of the NCR Century 200, with rod and core memories.

NCR CENTURY 200 TRUNK BANDWIDTH		
TRUNK	CORE MEMORY	ROD MEMORY
Low-Speed	120 KB	120 KB
1 High-Speed	487 KB	465 KB
2 High-Speed	909 KB	833 KB

● System Bandwidth

Input and output of data is concurrent with computing. The I/O control and the printer control interrupt the ALU and process each character on a memory cycle stealing basis, as explained earlier. The maximum I/O transfer rate of the NCR Century 200 is:

 ● 320 KB without the high-speed trunks
 ● 487 KB with 1 high-speed trunk
 ● 909 KB with 2 high-speed trunks

The rates mentioned on the preceding page are the theoretical maximum, and may not be attainable in specific instances because of cable delays, circuit speeds, and combinations of peripherals used.

Peripheral Types

NCR Century Series peripherals are classed as integrated peripherals or as freestanding peripherals.

● Integrated Peripherals

Four integrated peripherals are used with the NCR Century 200 system.

- ● Line printer
- ● I/O writer
- ● Console input device
- ● Punched card reader or punched tape reader

All integrated peripherals, with the exception of the printer, are physically attached to the processor. Each unit shares some of the logic and power supplies of the processor. An integrated unit does not delay the operation of any freestanding peripheral in the system; however, it will delay the operation of other integrated peripherals on the same trunk. Each of the integrated peripherals is assigned a definite trunk and position number. As previously stated, positions are the same for the integrated peripheral devices whether the system is octaplex or quadraplex.

● Freestanding Peripherals

A freestanding peripheral is one that may be connected to any I/O trunk at any position not reserved for integrated peripherals. A high-speed peripheral must be connected to a high-speed trunk; a low-speed unit may be connected to either a high-speed or a standard trunk. The only exceptions to this rule are devices that multiplex on a character basis. Such devices must be connected to a low-speed trunk.

Freestanding peripherals are described as level 1 or as level 2.

● Level 1 Peripherals

A level 1 peripheral occupies a position on a trunk and has its own control logic for processor communication. There may be up to eight level 1 units on one trunk.

● Level 2 Peripherals

Level 2 freestanding peripherals interface to the I/O trunk through a control unit or multiplexer. During I/O operation, the I/O control communicates with the control unit which supervises the actual peripheral operation by means of a second trunk similar in design to the I/O trunks.

The following illustration shows one level 1 peripheral connected directly to a trunk and three level 2 units connected to the same trunk and position number through the single level 1 control unit.

```
┌──────────────────────────────────────────────────────────────────┐
│              SIMPLIFIED LEVEL 1╱LEVEL 2 CONFIGURATION             │
├──────────────────────────────────────────────────────────────────┤
│  ┌──────────┬──────────┐                                         │
│  │          │          │                                         │
│  │  MEMORY  │          │                                         │
│  │          │          │                                         │
│  │          │   I/O    │   Level 2    ┌──┐─┌──┐─┌──┐             │
│  │          │  CONTROL │           ┌──┤  │ │  │ │  │             │
│  ├──────────┤          │   Level 1 ┌─┐                           │
│  │          │          │           └─┘─────────────────────       │
│  │          │          │                                         │
│  │          │          │   ● This trunk could be any trunk 1 thru 6 │
│  │   ALU    │          │   ● There can be up to 8 Level 1 devices on a trunk │
│  │          │          │   ● Each Level 1 peripheral could have its own Level 2 │
│  │          │          │     units                                │
│  │          │          │   ● Selection Trunk - picks proper trunk │
│  │          │          │               Position - picks Level 1 peripheral │
│  │          │          │               Unit - picks Level 2 peripheral │
│  └──────────┴──────────┘                                         │
└──────────────────────────────────────────────────────────────────┘
```
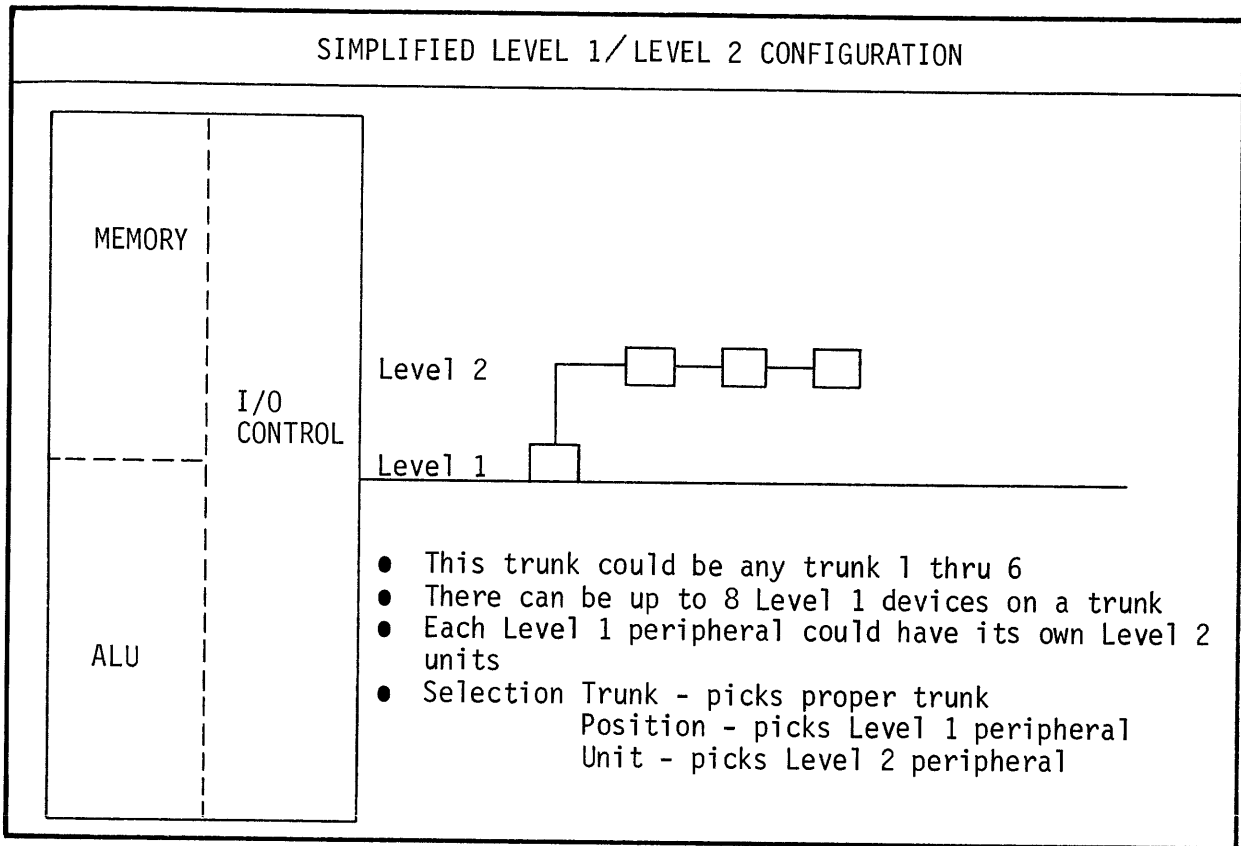
FUNCTIONAL OPERATION

The I/O operation is divided into three parts -- selection, data transfer, and termination -- which are discussed in this section. The interrupt indicator and the interrupt permit features are also discussed in this section.

Selection

An I/O operation is initiated when the ALU selects a peripheral in response to an INOUT command. Selection includes choosing the peripheral, initiating the desired function (read, write, print, etc.), and specifying any special instructions required by the specified peripheral.

PERIPHERAL SELECTION

I/O COMMAND INITIATED

ACCESS PAF CHARACTER AS SPECIFIED
BY "A" PORTION OF INOUT COMMAND

WAS CHARACTER RECEIVED BY PERIPHERAL

INCREMENT PAF ADDRESS BY 1

SEND PAF CHARACTER TO PERIPHERAL

IS THIS ALL THE PAF CHARACTER
REQUIRED BY THE PERIPHERAL

SET UP ADDRESS FOR S2 CHARACTER STORAGE
AS SPECIFIED BY "B" PORTION OF INOUT
COMMAND

STORE APPROPRIATE S2 CHARACTER

PROCESSOR TERMINATES I/O COMMAND
AND LEAVES I/O CONTROL IN CHARGE OF
DATA TRANSFER, PROVIDED SELECTION
WAS SUCCESSFUL

TERMINATE

● The Peripheral Address Field (PAF)

The effective A address of the INOUT command is the starting address of a memory location called the PAF. The PAF contains all information necessary (trunk and position character, unit character if desired, and function character) for completing an I/O selection operation.

Since some peripherals require more information than others, the PAF is a variable-length field. For example, a paper tape reader can only read; therefore, it is only necessary to select the reader for it to begin its function. Other peripherals, such as magnetic tape units, perform more than one function and must be selected and told what specific operation to perform. When the peripheral performs more than one function, the PAF must be longer than the minimum length of 1 character.

The first character of the PAF is the position character (P) as illustrated below.

P CHARACTER FORMAT							
b8	b7	b6	b5	b4	b3	b2	b1
0	T	T	T	0	P	P	P

TTT = I/O trunk number (0 through 7)

PPP = Position number (0 through 7)

In the NCR Century 200 System, b8 and b4 of the PAF P character are always 0 for compatibility between systems. The arrangement of PAF characters following the P character is determined by the functional characteristics of the peripheral. Level 1 peripherals that perform more than one function, but which need no additional addressing, require a function character following the P character. Control units require an additional character (unit character) in the PAF to select level 2 units connected to the control unit.

PAF FORMATS

ONE CHARACTER	TWO CHARACTER	THREE CHARACTER

ONE CHARACTER

BITS | O | T | T | T | O | P | P | P |
      | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

USED ON PERIPHERALS REQUIRING SELECTION ONLY.

POSITION ON A TRUNK

LEVEL 1 — CARD OR TAPE UNIT

1ST. CHARACTER NEEDED FOR SELECTION. NO FURTHER PAF NEEDED.

TWO CHARACTER

| P CHAR. | FUNCTION CHAR. |

USED FOR PERIPHERALS REQUIRING SELECTION AND FUNCTION CHARACTERS.

TRUNK

LEVEL 1 — PRINTER

1ST. CHARACTER FOR SELECTION.
2ND. CHARACTER FOR FUNCTION.

THREE CHARACTER

| P CHAR. | UNIT CHAR. | FUNCTION CHAR. |

USED FOR PERIPHERALS REQUIRING AN ADDITIONAL LEVEL OF SELECTION & FUNCTION CHARACTERS.

LEVEL 1 — CONTROLLER

LEVEL 2

1ST. CHARACTER FOR LEVEL 1 SELECTION.
2ND. CHARACTER FOR LEVEL 2 SELECTION.
3RD. CHARACTER FOR FUNCTION.
ADDITIONAL CHARACTERS IN PAF AS PERIPHERAL REQUIRES.

● **S2 Status Character**

The status character stored by the processor before the INOUT command is terminated is called the S2 status character. This character reflects the results of an attempt to select and activate the peripheral designated by the data in the PAF. The B portion of the INOUT command designates the memory location at which the S2 character is to be stored. Each selection attempt causes an S2 character to be generated.

Software checks the S2 character, and the processor functions according to decisions made in the software executive routines. Software alone is responsible for action taken after checking the S2 character.

The S2 character has four possible bit configurations:

● **Busy** (10000000)

The busy indication means that either the I/O trunk is tied up with another peripheral or that the selected peripheral itself is busy. However, freestanding control units for CRAM or disc can share seek time (the time required to locate the track where information is to be stored or read). Thus, the control unit for these peripherals can suppress the busy status and permit a seek operation even though another peripheral in the group is engaged in a seek, read, or write operation.

- <u>Standby</u> (10000010)

    This configuration is stored if the peripheral has been put in a standby condition.

- <u>Inoperative</u> (00000010)

    The inoperative S2 character is stored when the peripheral does not respond to the selection attempt or does not respond within the allotted time, but is not in the standby state.

- <u>Command Initiated</u> (01000000)

    This configuration is stored as soon as the selected peripheral has accepted all PAF characters. After storing the command initiated character, the ALU returns to internal processing operations, leaving the I/O control to supervise data transfer.

<u>Data Transfer</u>

During an output operation, each character is addressed, read out, and sent to the peripheral. During an input operation the character is received from the peripheral, memory is addressed, and the character is stored at this address.

When the selected peripheral is ready to receive or transmit a character of data, it sends a request for service to the processor. When this request for service is received, I/O control accesses memory to read out or store a data character.

```
+----------------------------------------------------------------------+
|                        I/O CONTROL OPERATION                          |
+----------------------------------------------------------------------+
|                                                                      |
|         O           START                                            |
|                                                                      |
|        [ ]          PERIPHERAL REQUESTS SERVICE,                     |
|                     SENDS RESPONSE NUMBER                            |
|                                                                      |
|        [ ]          I/O CONTROL  ACCESSES MEMORY                     |
|                                                                      |
|                                                                      |
|        [ ]          NA SUPPLIES THE DATA ADDRESS                     |
|                     TO BE USED                                       |
|                                                                      |
|        [ ]          INCREMENT NA BY 1                                |
|                                                                      |
|                                                                      |
|        [ ]          IF NA = TA, SET FLAG 1                           |
|                                                                      |
|                                                                      |
|        [ ]          IF OUTPUT MODE: READ OUT CHARACTER AND SEND TO   |
|                        PERIPHERAL                                    |
|                     IF INPUT MODE: STORE CHARACTER RECEIVED FROM     |
|                        PERIPHERAL                                    |
|                                                                      |
|        <>  ON        TEST FLAG 1                                     |
|       OFF  O         GO TO TERMINATION ROUTINE                       |
|                      THE I/O CONTROL WAITS FOR NEXT REQUEST FOR      |
|        [ ]              SERVICE                                      |
|                                                                      |
|         O           START                                           |
+----------------------------------------------------------------------+
```

This flow chart is a simplified description of I/O control operation during
data transfer. It is not intended as a programming guide.

Response Number

Each freestanding peripheral has a response number wired into its logic cir-
cuitry. Each time a peripheral requests service from the processor, the
response number accompanies the request. I/O control uses this number to com-
pute the starting address of the control word. Control word addresses begin
at memory location 1024. To calculate the CW address, the I/O control performs
the following steps:

1. In effect, multiplies the response number by 8.
2. Adds the results of the multiplication to 1024.

EXAMPLE:

To calculate the CW address when the response number is 5, I/O control does the following:

1. 5 x 8 = 40
2. 1024
    + 40
   ───────
   1064

Response number 5 indicates CW 5. The starting address of CW 5 is 1064.

A peripheral also has a trunk and position number; a level 2 peripheral also has a unit number. These numbers, used only for selection, are not related to the response number.

● Control Word

Control words, which begin at memory location 1024, consist of 2048 8-bit characters or 256 8-character words. The I/O control uses the CW data to address information, to determine when to terminate the operation, and to store status characters. There are two types of control words: one for the integrated printer and one for all other peripherals. (See publication ST-9402-42 for a discussion of the control word of the integrated printer.)

Each response number locates one control word. Since each peripheral has a unique response number, each peripheral has a unique control word.

STANDARD CONTROL WORD FORMAT					
S	NA			TA		2-CHARACTER AREA RESERVED FOR SOFTWARE USE
	N3	N2	N1	T2	T1	

S = A 1-character field where status is stored.
NA = A 3-character field containing the memory address of the next character to be accessed by the I/O control for output to the peripheral or the memory address where the next character received from the peripheral is to be stored. NA is incremented by 1 as each character is transferred.
TA = A 2-character field containing the terminating address. TA is compared to the N2 and N1 characters of the NA field to determine I/O completion. When NA = TA, data transfer is complete.

NA, which is used in both input and output operations is 16 bits in length
(2 characters) and provides the address into which the character is stored
or from which the character is read. NA is incremented by 1 and compared
to TA (2 characters in length), which always contains the last address plus
1 of the data storage areas. Comparing NA to TA determines when the process-
or has arrived at the end of a data field. If they are equal, the operation
terminates. This NA/TA arrangement allows a record size in the NCR Century
200 of up to 65,536 characters due to the 16-bit comparison of NA/TA.

The following illustration gives a general idea of NA/TA operation but does
not attempt to detail the actual internal processor flow.

READ OUT

READ NA
INCREMENT NA BY 1

READ TA

SET
TERMINATION   YES   NA = TA?
FLAG   NO

WRITE UPDATED NA

READ/WRITE DATA
        CHARACTER

GO TO INTERNAL PROCESSING
OR TOP OF FLOW FOR ANOTHER I/O FLOW
OR TERMINATION INTERRUPT FLOW

S3 STATUS CHARACTER TRANSMISSION

PROCESSOR

MEMORY

CONTROL WORD

S3

ALU

I/O
CONTR.
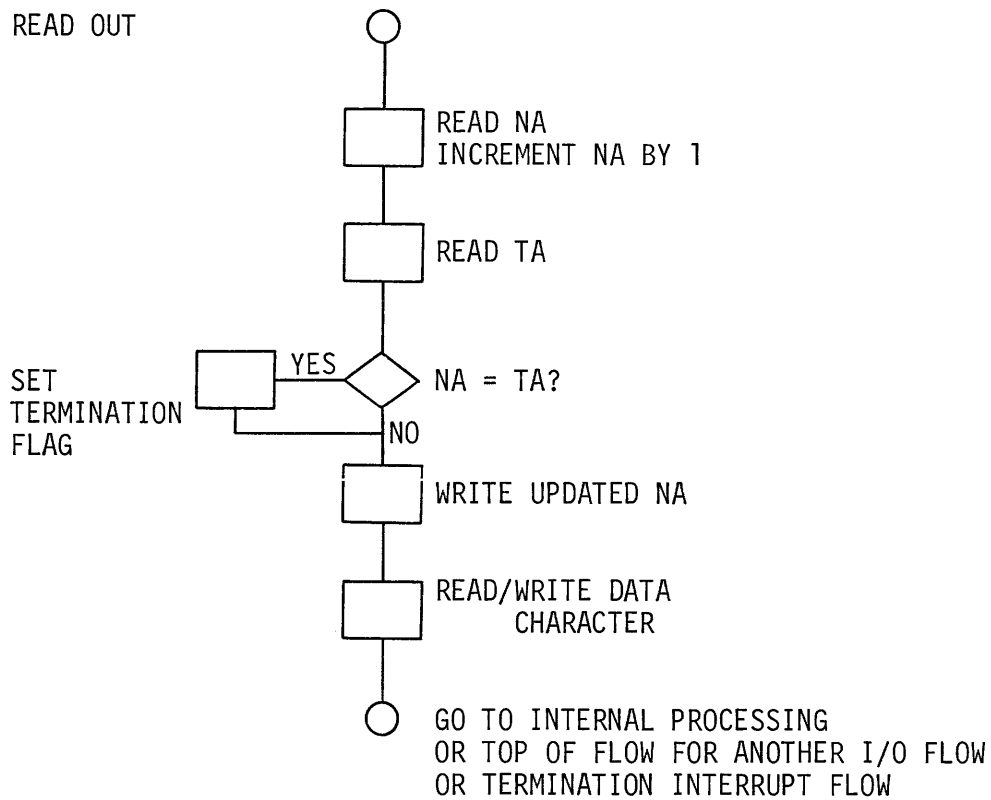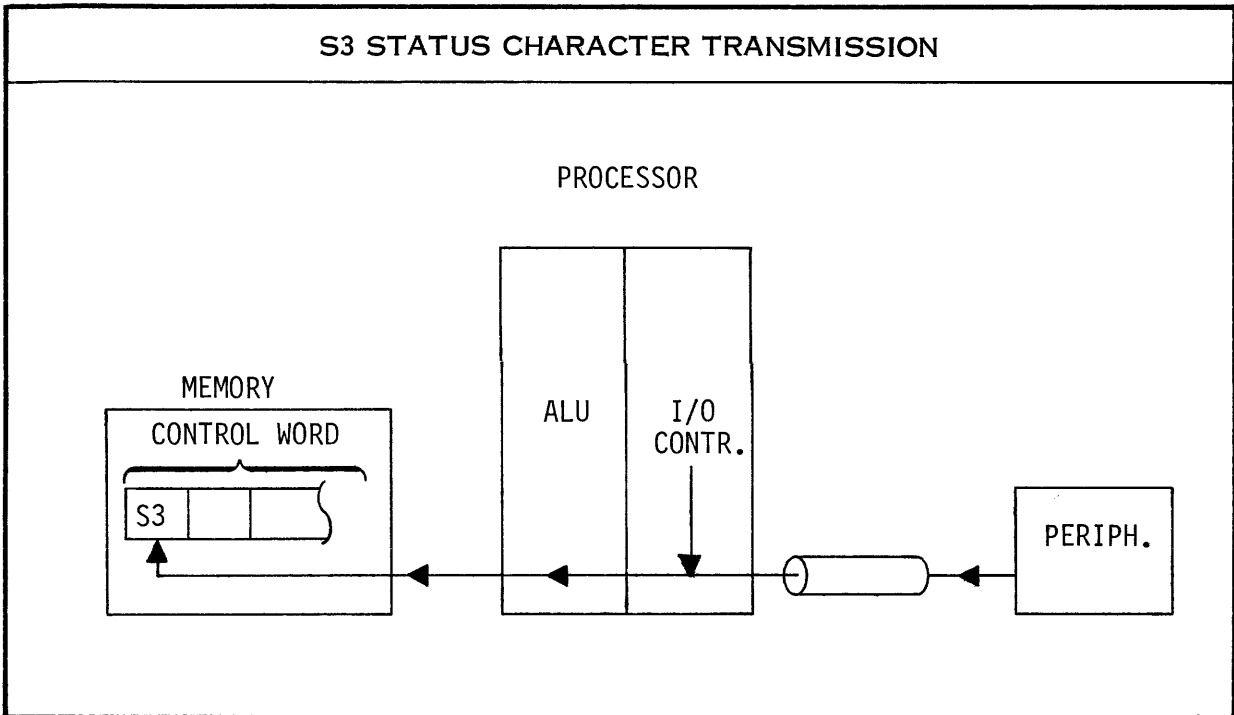
PERIPH.

Termination

Since NCR Century I/O operations occur offline after the peripheral has been successfully selected, status characters are used to inform the processor of what takes place during the operation. As previously explained, the processor generates and stores an S2 character to indicate the results of attempted peripheral selection. At termination of an I/O operation, a similar 8-bit character, called an S3 or S4 status character depending upon its origin, is stored in the first byte of the control word. This character reflects the outcome of the offline portion of the operation. System software examines this status character and takes the action its configuration dictates if the proper hardware indicator (see IP and II in this section) has been set ON.

Termination takes place when the I/O control determines that all required data has been transferred (NA = TA), or when the peripheral detects a terminating condition (End of Block, for example), or when an error occurs.

There are four types of termination.

● Normal Processor Termination

The processor terminate signal is sent to the peripheral by the I/O control when NA equals TA in the control word. In turn, the peripheral sends its response number and an S3 status character. Upon receipt of a processor terminate signal, a real-time, level 1 peripheral (such as a communications multiplexer) replies by sending an S3 segment complete status character to the I/O control. The processor must issue another read or write function, else a system overload occurs.

● Normal Peripheral Termination

If an I/O operation is complete (EOB is detected) before a processor ter-
minate is received, the peripheral requests service and sends a terminating
status signal and the appropriate S3 status character. As soon as the I/O
control takes charge from the processor, it stores the status character in
the control word and turns ON the interrupt indicator (II).

● Special Peripheral Termination

When the peripheral detects any of the following conditions, it immediately
sends a terminate status signal causing an S3 status character to be stored:

● Error when writing.
● System overload when writing.
● Write lockout when attempting writing, erasing, or rewinding.
   (The rewinding is initiated; writing and erasing are not.)
● Inoperative, when executing any function.
● Transmission errors.

Other conditions that are detected by the peripheral but are usually not
transmitted to the I/O control until an EOB is detected or until the next
INOUT command is initiated are:

● Error when reading (EOB).
● System overload when reading (EOB).

● Special Processor Termination

If the I/O control detects a transmission error during data transfer, it
sets the following sequence of operations in motion:

● The NA is stored in the control word.
● An error signal is sent to the peripheral or control unit.
● A special S4 status character is stored in the S byte of the control word.

S3 Status Character

At termination of an I/O operation, an 8-bit character, called an S3 status
character, is stored in the first byte of the control word. This character
reflects the outcome of the offline portion of the I/O operation. After pro-
cessor interrupt has occurred, system software examines this status character
and takes the appropriate action.

The S3 character has 10 possible configurations and meanings. If bit 8 and
bit 7 are both 0, the I/O operation is complete; if bit 8 and bit 7 are both
1, the transfer of one segment of data has been completed. If only bit 8 is
ON, the operation has been terminated early due to a transmission error or
operator intervention.

The following S3 definitions are general in nature. Certain peripherals have specific status characters that are explained in the separate publications dealing with these devices.

- Operation Complete (00XXXXXX)

  This configuration is stored if the I/O operation has been completed. Errors and exceptions encountered during the operation are indicated by various combinations of b6 through b1.

- Segment Complete (11XXXXXX)

  This configuration indicates processor termination occurred while a real time peripheral had more data to transmit to its control unit. The processor must then send a read or write function to activate the remote peripheral. If a data character arrives at the control unit before the read or write function code, a program overload occurs.

- Error (00100000)

  This configuration occurs when the selected peripheral detects an error (normally a parity error) during an I/O operation. If the error is detected while the peripheral is performing a read operation, it is noted and sent as a bit in the S3 status character when terminating status is sent. If the error is detected while writing, a terminating status signal and the proper S3 status character are sent to the processor immediately.

- System Overload (00010000)

  When the I/O control does not respond to the selected peripheral's request for service within its character time, the unit detects a system overload. Character time, the amount of time required for a peripheral to receive or transmit 1 byte of data, varies with the peripheral unit. Data transmission ceases when a system overload is detected.

  If a system overload is detected during a write operation, a terminating signal and the appropriate S3 status character are sent to the processor immediately. When a system overload is detected during a read operation, the peripheral notes the condition and sets the proper bit in the S3 status character when a terminating status is sent.

- Media (00001000)

  This configuration is stored when the selected peripheral detects a warning marker, such as a magnetic tape destination warning marker, during a write operation. The warning mark is noted and sent as a bit in b4 of the S3 status character when the transmission signal is sent. The I/O control continues data transmission even though the peripheral has detected the marker.

● Write Lockout (00000100)

This configuration is stored when the peripheral attempts to write but is in the write lockout state. The elapsed time between S2 and S3 storage in this case may be so slight as to be undetectable by the program.

● Inoperative (00000010)

The inoperative configuration is stored when a malfunction is detected by the peripheral after it has been activated. Data transmission ceases immediately and the terminating status signal is sent along with the status character.

● Special (00000001)

This configuration is stored to indicate any condition not included above. The actual configuration used will depend upon the specific peripheral involved.

● Transmission Error (10000001)

This configuration is stored if a transmission parity failure is detected by a peripheral during the I/O operation. When a parity error is detected, the peripheral immediately deselects itself and sends the character.

● Standby (10000010)

This configuration is stored when the peripheral is found to be in the standby state.

During an I/O operation, more than one S3 condition can occur in some units. The peripheral relays all conditions, which are reflected in the status character, to the I/O control when the operation terminates.

S4 Status Characters

The I/O control generates and stores an S4 status character in the control word when either the I/O control or the printer control detects an error condition. This error condition also causes the I/O control to notify the peripheral to stop transferring data, to inhibit sending an S3 status character, and to deselect.

There are seven S4 status characters: three primary status characters and four combination status characters that are multiples of the primary status characters. The following illustration contains all S4 status characters that may be stored.

S4 STATUS CHARACTERS	
STATUS CONDITION	BIT CONFIGURATION
Transmission Error (TE)	10000001
Latent Memory Error (LME)	10000100
TE and LME	10000101
Latent Program Error (LPE)	10001000
TE and LPE	10001001
LME and LPE	10001100
TE and LME and LPE	10001101

The conditions that cause an S4 status character to be stored and the primary S4 status characters are described next.

● Transmission Error (10000001)

The transmission error status bit configuration is stored if a transmission parity failure is detected by the I/O control or the printer control. The I/O control signals the peripheral to stop sending data, to inhibit sending S3 status, and to deselect.

● Latent Memory Error (10000100)

Any memory error (ME) detected by the I/O control or the printer control is a latent ME. If such a condition occurs, the configuration for latent ME is stored and the I/O control signals the peripheral to stop sending data, to inhibit sending S3 status, and to deselect.

A latent ME is detected during an I/O operation when the I/O control attempts to read NA or TA or to read a data character from memory. In both cases, the S4 status character (10000100) is stored in the associated control word.

In the case of an ME when reading NA or TA, the NA is not updated and the character is not transferred to the peripheral or accepted by the processor from the peripheral.

When an ME occurs while reading a data character from memory, the NA is updated, but the character is not transferred to the peripheral.

● Latent Program Error (10001000)

Any program error (PE) detected by the I/O control or printer control is a latent program error. If such a condition occurs, the latent PE configuration is stored and the I/O Control signals the peripheral to stop sending data, to inhibit sending S3 status, and to deselect.

A latent PE is caused by the printer control or by NA being greater than memory size. In the first case, the proper S4 status (10001000) is stored in the printer CW (1032) and the I/O operation terminates after sending a processor terminate to the printer.

When the NA is greater than memory size, termination is determined by the size of memory. If memory size is 64K or 512K, the NA is not updated, i.e., it is left pointing at the last memory location, and the last memory location is left undisturbed. When memory size is not 64K or 512K, NA is updated, i.e., left pointing at 1 more than actual memory size, and the data character is neither accepted nor output by the processor.

Regardless of the cause of a latent PE, the S4 status character (10001000) is stored in the associated CW and a processor terminate is sent to the peripheral.

Interrupt Permit and Interrupt Indicator

An interrupt indicator is set ON by a peripheral termination and is set OFF by the interrupt trapping flow when program interrupt occurs. The interrupt permit must be turned ON before a peripheral terminating status signal can be recognized and the terminating trap routine entered. Software normally sets the IP ON at the proper time.

IP is set ON in the system executive program flow so that the processor can be interrupted and the interrupt routine performed when the peripheral terminates. II is the means by which the processor remembers that it has been interrupted. The program is interrupted and the trap routine is entered. When the trap routine is entered, II and IP are turned OFF. II and IP have hardware indicators on the console panel of the processor. A functional description of the interrupt trapping flow may be found under "Between Commands Testing", in this publication.

- ### Interrupt Permit

  The NCR Century 200 System has two commands to control IP: SET IP ON and SET IP OFF. The SET IP ON turns the interrupt permit ON. The next command to be performed will be located at the address specified by the A portion of the command. If A is not a legal command address, a PE occurs and the control register is left undisturbed. The T and B portions of the command are not used.

  SET IP OFF turns the interrupt permit OFF. Everything else functions exactly the same as in the SET IP ON command.

  The IP provides the capability of interrupting the program flow in order to enable a trap routine to process the termination of an I/O operation. The interrupt to the trap routine is orderly, occurring between command executions. When processing is complete, the trap routine exits to the program where the interrupt occurred, enabling the processor to resume the program at the point it was interrupted.

● <u>Interrupt Indicator</u>

The interrupt indicator is used to remember that an interrupt situation
has occurred, until such time as program interruption can be initiated.
An interrupt situation occurs when the I/O control or printer control de-
tects a terminating status signal or when a latent error condition occurs.

OPTIONS

INTRODUCTION

Several options are available for the NCR Century 200 System. Any of these options may be installed at the user's site:

- Trace
- Extended simultaneity
- High-speed trunks
- Multiprogramming
- Interval Timer
- Extended memory
- 315 Simulator
- 1401 Simulator
- Floating point

TRACE OPTION

The trace option provides the means to monitor each command execution. Included in the option are a trace permit indicator and three additional machine commands. The three commands are STORE TRACE, LOAD TRACE, and LOAD MONITOR REGISTER. These commands are explained in the "NCR Century 200 Hardware Commands and Command Timing," under this tab.

Trace Execution

When the programmer turns the trace permit indicator (TP) ON in the program, the processor, upon completion of each command execution, enters the BCT. As a result of BCT, the hardware performs the following functions:

- The state of the processor is stored in the program status word (location 0052-0063).
- The overflow flag (OF), the repeat indicator (RI), and the trace permit are turned OFF.
- The S flag is turned ON (if multiprogramming option is included).
- The control register (CR) is loaded with the contents of locations 0277-0279.

The hardware functions of BCT for trace are then complete and the processor will enter the trace trap routine.

If a memory error (ME) occurs during the readout of memory locations 0277, 0278, and 0279, the ME indicator is turned ON, and the control register remains undisturbed. The subsequent memory error trap stores the same control register contents as did the trace trap.

If locations 0277, 0278, 0279 contain an illegal address (greater than memory size or not evenly divisible by 4), the control register is loaded with the illegal address. The subsequent program error trap stores this illegal address as the contents of the control register.

Trace with Monitor

When the trace option is used with the MONITOR SWITCH, the processor has the means of checking a specific memory location. The memory location is specified by the programmer in the LOAD MONITOR REGISTER command. The command loads a memory address into the monitor register and, when the MONITOR SWITCH is ON, TP will be turned ON when the specified memory location is written into. With TP ON, when the current command is executed, the processor will begin BCT and proceed as previously explained for trace execution.

When using trace with monitor, the TP indicator is not turned ON under the following conditions:

- The processor is in a hardware trapping flow.
- The monitored memory location is written into by an INPUT/OUTPUT flow.

Monitor Register

The monitor register is used to contain a memory address that is monitored for writing. If the memory location specified by the monitor register is written into, the TP indicator is turned ON. The monitor register may be loaded by command (LOAD MONITOR REGISTER) or from the console address switches when the FUNCTION SELECT SWITCH is in the MONITOR position. In systems with the multi-programming option, the address in the register is compared to a relative address in memory.

When the MONITOR switch is OFF, the monitor register has no effect on the processor.

The following illustration shows the monitor register and its relation to other registers in the processor.

MONITOR REGISTER
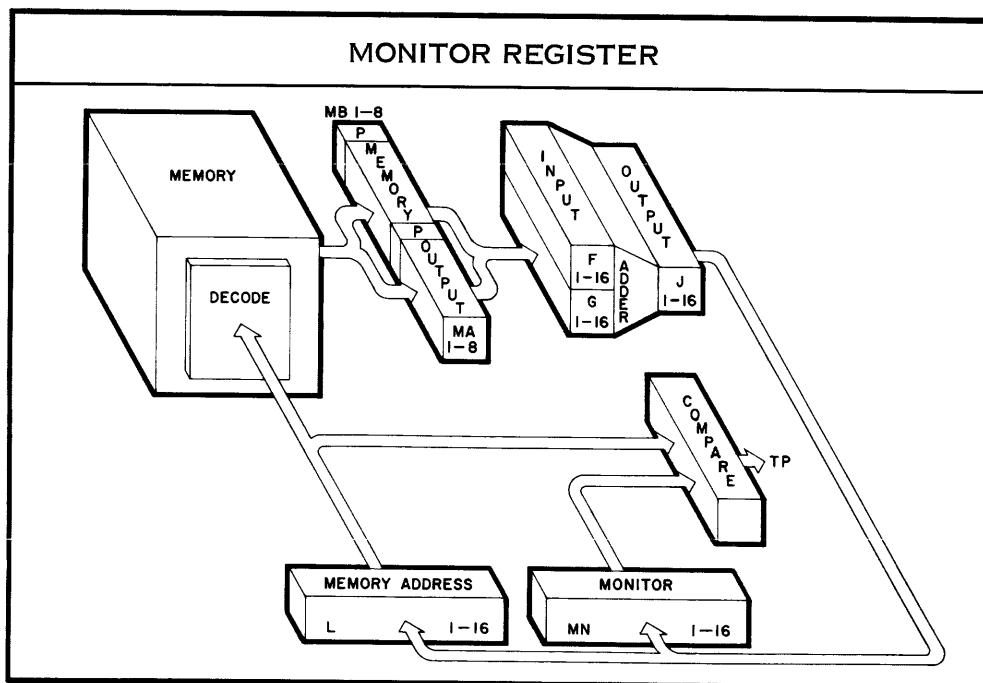
EXTENDED SIMULTANEITY OPTION

The extended simultaneity option upgrades a quadraplex (4-trunk) system to an octaplex (8-trunk) system. The upgrading may be accomplished in the field and includes an expanded I/O control section with four additional common trunk interfaces.

The upgraded system has the capability of handling eight simultaneous I/O operations (one on each trunk), in addition to computing operations.

When the extended simultaneity option is included in a system, the high-speed trunk option is included for trunks 6 and 5. (See I/O control section for trunk and position assignments.)

HIGH-SPEED TRUNK OPTION

The high-speed trunk option is used to increase the bandwidth of an I/O trunk, which also increases the bandwidth of the system. A maximum of two trunks can be upgraded to high-speed trunks. This upgrading may be accomplished in the field.

Only the highest priority (highest numbered) trunk(s) can be high-speed trunk(s). In systems having the extended simultaneity option, trunks 6 and 5 must be high speed trunks.

A high-speed trunk is also required for the operation of high-speed peripherals. A high-speed peripheral is one whose maximum transfer rate is 120KB or greater.

A high-speed trunk can accommodate both high-speed and conventional peripherals. However, data cannot be multiplexed on a character basis over a high-speed trunk.

System I/O Bandwidth

The high-speed trunks raise the system I/O bandwidth as follows:

- One trunk operating -- 487 KB
- Two trunks operating simultaneously -- 909KB

Functional Operation

A high-speed trunk uses live registers for NA and TA of the control word. After a read or write function has been initiated and before the first character is transmitted, the high-speed peripheral sends a ready-to-transmit signal. The I/O control, upon receiving this signal, accesses the appropriate control word and transfers NA and TA to live registers.

If no ready-to-transmit signal is received from the peripheral, it is not considered to be a high-speed peripheral. NA and TA are set up during the first data transfer. The trunk then runs in the high-speed mode.

The high-speed trunk control is designed to wait after receipt of a character from a high-speed peripheral. The time of the wait period is used to permit another request for service from the trunk just serviced.

If the device being serviced is a high-speed peripheral, then only another high-speed trunk or the integrated printer can be serviced during the wait period. If the device being serviced is not a high-speed peripheral, then any other trunk or the integrated printer can be serviced during the wait time.

MULTIPROGRAMMING OPTION

The multiprogramming option provides for several programs to share processor and memory, with complete memory and peripheral protection, under control of a supervisor program. Also included in this option is program relocatability, with each program assuming a starting location of zero, using a set of independent index registers.

This option, which can be added in the field, consists of minimum memory size of 64K, additional hardware commands (LOAD BAR and LOGIC), additional command functions, control and compare logic, supervisor/user state (determined by the S flag), and the interval timer.

The multiprogramming option includes the extended simultaneity option and may be physically installed on a base system (32,768 bytes). Due to the memory requirements of the supervisor program (system software) and the user programs, it will normally be installed with, or added to, a system containing the extended memory option.

User and Supervisor States

The processor is always in one of two states – user or supervisor – indicated by the status of the S flag. When the S flag is OFF, the processor is in the user state; when the S flag is ON, the processor is in the supervisor state. The S flag is turned ON during the following flows:

- Program Interrupt (Peripheral Terminate)
- Console Loading (LOAD Button)
- Command Code Trap
- Program Error
- Memory Error
- Trace

The S flag is turned OFF only during the execution of a RESTORE command, provided the pertinent bit of the referenced status word so specifies.

Base Address Register (BAR) and Limit Address Register (LAR)

The system software, through the use of the hardware registers BAR/LAR, provides the following features.

User programs may be written assuming a starting location of zero, relieving the programmer of the burden of providing absolute memory locations to ensure data protection. The system software, through the use of BAR, will locate and allow execution of the user program in the most efficient manner. This provision for program relocatability also allows each user program to have its own set of 63 index registers.

Memory protection is provided through the use of LAR. Since a memory access of addresses greater than the contents of LAR is not permitted, one user's program is not allowed access to another user's program.

Each register is 7 bits in length (8 bits in systems that have the extended memory option) and contains an address in 2048 character increments. That is, the contents are used as through they had eleven 0 bits appended to the right.

EXAMPLE: BAR OR LAR   "IMAGINARY" BITS

                00000001   00000000000

Decimal equivalent address = 2048

When BAR/LAR is used, the comparison between LAR and the effective address is always performed before the BAR addition is accomplished. The contents of LAR also assume a starting location of zero as does the user's program. When the contents of LAR are equal to zero, no upper limit to memory accesses is specified. If the contents of LAR are equal to or less than bits 12-16 of the effective address, an immediate PE occurs. Therefore, to have a good LAR comparison, the contents of LAR must be greater than bits 12-16 of the effective address.

If the LAR comparison is satisfactory, the contents of BAR are added to bits 12-16 of the effective address just prior to the memory access. When BAR is used, the program and its functions are executed relative to the address contained in BAR. If a carry out of the 16th bit position (19th bit on systems with the extended memory option) occurs during the BAR addition, an immediate PE results. If the contents of BAR are equal to zero, no base address is specified.

BAR/LAR - User State

When the processor is in the user state (S flag OFF) command addressing, setup, and execution are subject to BAR/LAR processing. The contents of the sequence control register (CR) are transferred to the memory address register (L). Bits 12-16 of the L register are compared to LAR, and if the comparison is satisfactory, BAR is added to bits 12-16 of the L register for command access.

● Command Setup and Execution - No Indexing

When command setup with no indexing specified is completed, the A2A1 characters form the effective A operand address, which is stored in the LA register. The contents of the LA register are transferred to the L register, and bits 12-16 are compared to LAR. Assuming a satisfactory comparison is made, BAR is added to bits 12-16 of the L register and the A operand is read from memory.

EXAMPLE:

(Addresses shown in decimal notation for simplification.)

1.

Q	RA	A2	A1
	0	0	8000

MODE

NO INDEX
REGISTER
SPECIFIED

COMPARE ①   LAR 12-19   10240

<

Bits 12 - 19 of the effective A
operand address are less than
bits 12 - 19 of LAR.

②

EFFECTIVE A OPERAND
ADDRESS BEFORE BAR
PROCESSING

| 8000 |
| 16384 |
| 24384 |

EFFECTIVE A OPERAND
ADDRESS AFTER BAR
PROCESSING

② ← BAR 12-19   16384

2.

Q	RA	A2	A1
	0	0	12288

COMPARE ①   LAR 12-19   10240

>

PE
Bits 12 - 19 of the effective A
operand address are greater than
bits 12 - 19 of LAR.

3.

Q	RA	A2	A1
	0	0	8192

COMPARE ①   LAR 12-19   8192

=

PE
Bits 12 - 19 of the effective A
operand address are equal to
bits 12 - 19 of LAR.

NOTE

These procedures apply to the B operand also.

● Command Setup and Execution – Indexing Specified

In command setup with indexing specified, the RA character is the address of the index register. Just prior to the memory access of the index register, the BAR contents are added to the RA character. This memory access is subject to LAR verification. The contents of this user index register are added to the A2A1 character to form an effective address. The command execution phase is entered and after LAR has been checked, the effective address is added to BAR to obtain the A operand address.

EXAMPLE:

(Addresses shown in decimal notation for simplification. These procedures apply to the B operand address also.)

Q	RA		A2	A1
	27	0	10000	

└─ MODE

ADDRESS OF INDEX REGISTER WORD 27
0108

COMPARE ① <

LAR 12 - 19
16384

②

0108 + 32768
32876

②

BAR 12 - 19
32768

32876	32877	32878	32879
		5000	

USER INDEX REGISTER WORD

③

10000 + 5000
15000

EFFECTIVE A OPERAND
ADDRESS BAR PROCESSING
IS COMPLETE

COMPARE ④ <

15000

LAR 12 - 19
16384

⑤

15000 + 32768
47768

⑤

BAR 12 - 19
32768

EFFECTIVE A OPERAND ADDRESS AFTER BAR PROCESSING
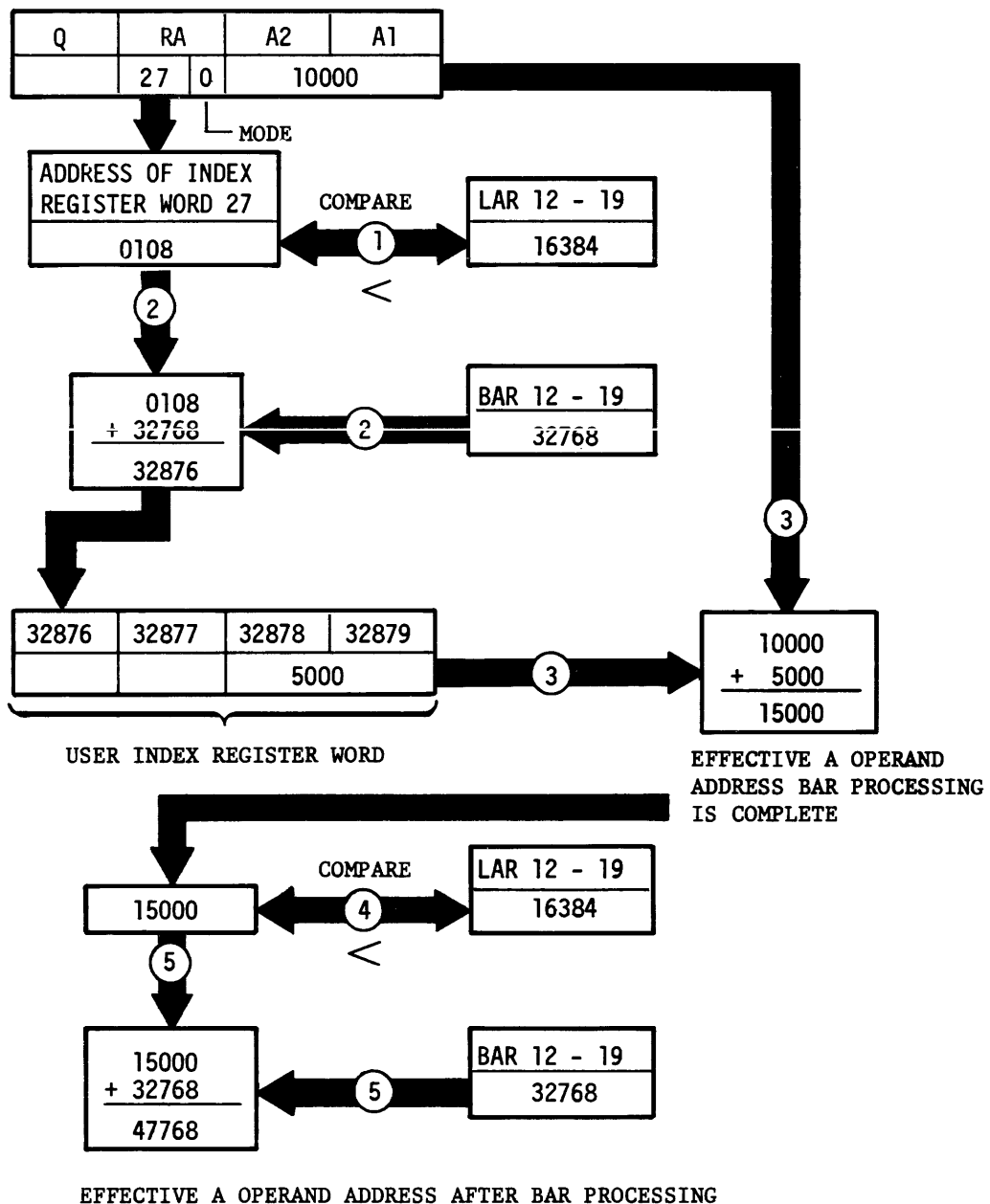
BAR/LAR processing in the user state is effective during command setup in all modes of addressing. All intermediate fields referenced during modes 1 and 2 are those relative to the contents of BAR.

BAR/LAR - Supervisor State

When the processor is in the supervisor state (S flag ON), command addressing is not subject to BAR/LAR processing. In command setup and execution, two hardware flags, one pertinent to the A value (flag A) and the other pertinent to the B value (flag B), indicate whether or not the BAR and LAR are effective.

Flag A is set ON or OFF during the setup phase dealing with the first four characters of the command. If the RA character from the command references any of the index registers 1 through 31, flag A is set ON; otherwise, flag A is set OFF. The index register referenced when flag A is ON is the user index register relative to the current BAR. RA characters obtained by mode 1 indirect addressing have no effect on flag A.

Flag B is set ON or OFF during the setup phase dealing with the last four characters of an 8-character command. If the RB character from the command references any of the index registers 1 through 31, flag B is set ON; otherwise, flag B is set OFF. The index register referenced when flag B is ON is the user index register relative to the current BAR. RB characters obtained by mode 1 indirect addressing have no effect on flag B.

Whenever flag A is ON, all memory accesses required to obtain the A operand, including those resulting from indirect addressing, are subject to BAR/LAR processing. Flag B functions in the same manner for the B operand. BAR/LAR processing does not occur during A operand addressing if flag A is OFF or for B operand addressing if flag B is OFF.

The following special locations, although they are within the index register area and are frequently used interchangeably with the index registers, are not dependant on flag A or flag B in the supervisor state. The absolute index register (as with BAR = 0) is referenced in these cases:

- The repeat counter in IR8 referenced by the REPEAT command.
- The jump link register in IR8 stored by the JUMP command.
- The next address index register in IR9 stored by the DECODE and SCAN commands.
- The count counter in IR16 referenced by the COUNT command.

LAR - Additional Function

The following commands may cause a transfer of control to the A address.

- BRANCH (8 commands)
- TEST CHARACTER EQUAL
- TEST CHARACTER UNEQUAL
- TEST BIT
- COUNT
- JUMP
- SET IP ON
- SET IP OFF

When LAR is used and any of the above commands is executed the A address is compared to the contents of LAR before it is transferred to the control register. If the A address is greater than the contents of LAR and the conditions that cause a transfer of control are satisfied, an immediate PE occurs and the control register is undisturbed. If the A address is smaller than the contents of LAR or if LAR equals 0, the command proceeds and is tested as specified in the NCR Century 200 commands publication.

BAR - Additional Function

When the processor is in the user state (S flag OFF), trapping flows will cause the error status words, program status words, and error code characters to be stored relative to the contents of BAR.

BAR/LAR FUNCTIONAL FLOW
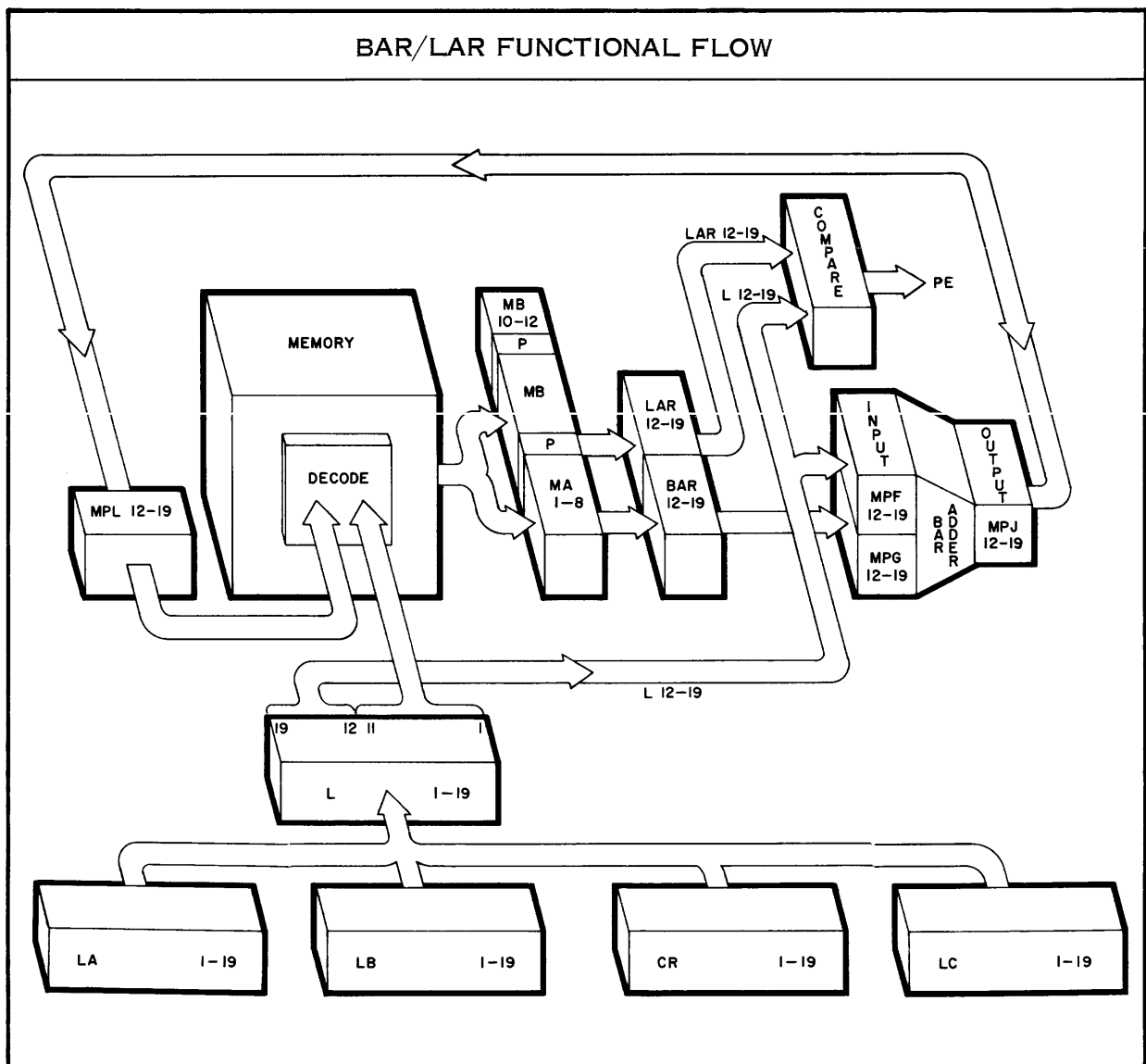
Privileged Commands

A privileged command is one which can only be executed when the processor is in the supervisor state. Certain of these commands (marked with * below) provide peripheral protection, while others (marked with **) are used by system software to establish program priority.

    INOUT*
    SET IP ON*
    SET IP OFF*
    SWITCHES INPUT**
    RESTORE**
    WAIT
    LOAD BAR**
    LOAD MONITOR REGISTER**
    EXECUTE**

An attempt to execute any of these commands when the processor is in the user state causes a command code trap.

● The RESTORE Command

The RESTORE command, being privileged, can only be executed when the processor is in the supervisor state. The A-operand address specifies the location of a 12-character status word. The condition of bit 8 of the ninth character determines the S flag state after execution. If bit 8 is OFF, then the S flag is set OFF; if bit 8 is ON, the S flag remains ON. All memory addressing during the execution of this command is dependent upon the original state of the S flag.

When a RESTORE command is to return control to the user state, the address that is to be transferred to the control register is compared to the contents of LAR. If the address is greater than LAR or equal to LAR, an immediate PE occurs and the control register remains undisturbed. If the address is less than LAR, or if LAR is equal to 0, the command is executed as described in "NCR Century 200 Hardware Commands and Command Timing," under this tab.

● Status Word

All the trapping flows store the state of the S flag in bit 8 of the ninth character of the status word. The state of the S flag stored is the state prior to any change which may occur in the flow. The remaining 7 bits of the ninth character are:

    ● b7   Always OFF
    ● b6   Overflow flag
    ● b5   Repeat Indicator
    ● b4   Trace flag (Trace Option)
    ● b3   Greater flag
    ● b2   Equal flag
    ● b1   Less flag

The remaining characters of the status word are explained in the memory section of this manual and shown on the memory map.

● The LOAD BAR Command

To execute the LOAD BAR command, the processor must be in the supervisor state and the A operand address must be evenly divisible by 4, or a PE results. If this happens, the contents of BAR and LAR remain undisturbed. The A operand address specifies a 4-character field, of which only the two rightmost characters are used. Of these two characters, the left character specifies the value to be loaded into the BAR, and the right character specifies the value to be loaded into the LAR. Only the five low order bits of each character are used, unless the extended memory option is incorporated; then, all 8 bits of each character are used.

Additional Console Features

The console is described in the operator's console section of this manual. However, some additional features are incorporated in the console with this option, and are explained as follows:

When the HALT switch is ON, addresses entered from the console do not have BAR added to them under any circumstances.

NOTE

Manual operation from the console permits loading, entering and displaying addresses and data in hexadecimal notation according to the switches on the console. Should the program be stopped, however, by turning on the halt switch or by executing a WAIT command, then the address displayed will be relative. Where the S flag is ON, this is the absolute address in memory and data may be entered or displayed directly, if it is so desired, regardless of the state of the A or B flags. In the case where the S flag is OFF, the address displayed is the address in the user's program and the contents of BAR must be added to it when it is necessary to manually enter corrections or display data. Any addresses changed for this purpose must be reset before restarting the user program. Except for initial loading from the COT, it is unlikely that console manipulation will be permitted in a multiprogramming system.

The LOAD button, in addition to its other functions, turns ON the S flag, which is displayed on the console. Also included in the option is the ability to display the BAR/LAR contents and also the five or seven significant memory address register bits.

INTERVAL TIMER

A timer is provided as an option to permit programs to cause an interrupt after a specified number of 1 millisecond increments. The timer makes use of certain input-output features in its operation. The NA portion of the Control Word is counted up by one every 1 millisecond until NA=TA. When NA equals TA an S3 status (operation complete) is stored and the interrupt indicator is set ON. The timer never stops, except in Halt.

Control Word

The timer uses a control word at location 336. The format of this control word is identical to that of other control words.

Status

Three S3 status characters are stored by the I/O Control for the Interval Timer:

- Operation Complete (0000 0000)
- System Overload   (0001 0000)
- Special   (0000 0001)

Operation complete is stored every time NA=TA.

System overload is stored if a "tic" (a count of the timer) is missed by the I/O Control because it is too busy to service it. Each "tic" of the timer causes a channel request to be sent to the I/O Control for 750 us. Once a "tic" times out without being serviced, any subsequent channel request serviced causes a system overload to be stored in the control word.

The special S3 is stored whenever the machine goes into the Halt state.

Operation complete and system overload both set the interrupt indicator ON.

A latent ME will cause an S4 status character (1000 0000) to be stored and the interrupt indicator to be set ON.

Accuracy

The timer has a basic increment of 1 millisecond $\pm$ .01%. The timer counts NA up to 65535 and then starts over again at 00000. Therefore, the maximum time between interrupts caused by NA=TA, is 65536 milliseconds or 65.536 seconds.

Halt

The timer runs continuously, except when the CPU is put into the Halt state. When that happens, the special S3 status will be stored and the timer will stop counting NA. The timer will resume the count when the CPU is taken out of the Halt state.

EXTENDED MEMORY OPTION

The extended memory option provides for memory sizes larger than 65,536 characters. The size of memory may be upgraded or downgraded in the field. The following memory sizes are available with the Century 200 System.

- 32,768 characters ⎱
- 49,152 characters ⎰   with any NCR Century 200 system
- 65,536 characters ⎰

- 98,304 characters
- 131,072 characters
- 196,608 characters
- 262,144 characters
- 393,216 characters
- 524,288 characters

with any NCR Century 200 system having the extended memory option

Addressing

Addressing of upper memory locations has been provided for, through the use of 19-bit index registers (refer to the Memory section of this publication).

An index register must be used when specifying an address higher than 65,535, since this is the maximum value of a 16-bit partial address (A or B portion of a command).

When indexing, the partial address is added to the contents of an index register (designated by RA or RB of the command) to produce an effective address of a field in memory. The resulting effective address may be greater or less than the address originally contained in the index register, depending on whether the value assigned to the partial address causes a positive or negative displacement. In the following discussion, negative displacement is referred to as decrementation, and positive displacement as incrementation.

The mode of addressing specified will determine whether or not the actual contents of the index register are changed (refer to the ALU section of this publication).

- Decrementation without the Extended Memory Option

    When indexing in systems without the extended memory option the 16 bits of the partial address are binarily added to the 16 bits contained in the index register to form the effective address. Carries are propagated up to the sixteenth bit; carries past the sixteenth bit are ignored.

    A value of 65,536 (binary value of a seventeenth bit position) is needed to cause a carry past the 16th bit of the adder. However, since this carry is ignored, 65,536 could be lost when adding. It is the loss of this amount when adding to form an effective address that allows decrementation to occur.

    Decrementation of the address contained in the index register is specified by making the partial address equal to: 65,536 minus the displacement value.

    EXAMPLE:

    Assume that the address contained in an index register is 300 and the effective address required is 100.

- Subtraction made by programmer to determine what _partial address_ to use in the command:

65,536	Value needed to cause a carry past the 16th bit
- 200	Displacement value, equal to: index register contents of 300 minus effective address of 100
65,336	Partial address

- Operation accomplished by the adder when indexing:

65,336	Partial address
+ 300	Contents of the index register
65,636	Adder did not actually reach this value, since a carry past the 16th bit resulted during the addition.
- 65,536	Value lost by the carry past the 16th bit being ignored.
100	Effective address – the desired–200 displacement from the address of 300 contained in the index register

	32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1
	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Index register contents =   300 =   0 0 0 0   0 0 0 1   0 0 1 0   1 1 0 0

Partial address =   65,336 =   1 1 1 1   1 1 1 1   0 0 1 1   1 0 0 0

Effective address =   100 = 1   0 0 0 0   0 0 0 0   0 1 1 0   0 1 0 0

↑
This carry ignored.

- Decrementation with the Extended Memory Option

When indexing in systems with the extended memory option, the 16 bits of the partial address are binarily added to the contents of the index register to form the effective address on such systems. The index register is expanded to include 19 bits, and carries are propagated up to the nineteenth bit. A carry past the nineteenth bit is ignored.

A value of 524,288 (binary value of a 20th bit position) is needed to cause a carry past the 19th bit of the adder. However, since this carry is ignored, 524,288 could be "lost" in the addition being performed.

The partial address contains only 16 bits. When these bits are placed into the adder, the state of b16 of the partial address determines the states of the adder input bits 17, 18 and 19. Bit 16 of the partial address must be equal to 1 if decrementation is to occur. Adder input bits 17, 18, and 19 will then be set to 1's automatically for a value of 458,752 (binary value of bit positions 17, 18, and 19). This value 458,752 is "in addition" to the value of the 16-bit partial address put into the adder.

During the addition of the index register contents to the 16-bit partial address, a value of 65,536 plus the set 458,752 causes a carry past the 19th bit. In other words, the additional 65,536 needed to cause a carry past b19 is the same value needed to cause a carry past b16 in systems without the extended memory option. Because of this, decrementation can be specified in the same manner as in systems without the extended memory option provided b16 of the partial address is equal to 1. Decrementation of the address contained in the index register is specified by making the partial address equal to 65,536 minus the displacement value.

EXAMPLE:

Assume that the address contained in an index register is 300 and the effective address required is 100.

- Subtraction made by programmer to determine what partial address to use in the command:

65,536	Value needed to cause a carry past the 19th bit when adder input bits 17, 18, and 19 are set to 1's.
− 200	Displacement value, equal to: index register contents of 300 minus effective address of 100
65,336	Partial address

- Operation accomplished by the adder when indexing:

65,336	Partial address
+ 458,752	Binary value of bit positions 17, 18, and 19. (Set to 1's by bit 16 of the partial address being equal to 1.)
524,088	
+ 300	Contents of the index register.
524,388	Adder did not actually reach this value, as a carry past the 19th bit resulted during this addition.
− 524,288	Value "lost" by the carry past the 19th bit being ignored.
100	Effective address

(Note that this value is the desired −200 displacement from the address of 300 contained in the index register.)

262,144	131,072	65,536	32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Index register contents = 300 =   0 0 0   0 0 0 0   0 0 0 1   0 0 1 0   1 1 0 0

Partial address =   65,336 =   1 1 1   1 1 1 1   1 1 1 1   0 0 1 1   1 0 0 0

                         Set to 1's by 16th bit

Effective address =   100 = 1   0 0 0   0 0 0 0   0 0 0 0   0 1 1 0   0 1 0 0

            This carry ignored.   Carry propagated between 16th and 17th bits.

Decrementation from one memory group of 65,536 characters to a next lower address group can be accomplished as illustrated in the following example.

Assume that the index register is pointing to address 131,073 which is located in the third group of 65,536 characters. After decrementation of 10 the effective address is in the second group of 65,536 characters.

EXAMPLE:

(Displacement value = 10)

Partial address = 65,536 minus the displacement value

           = 65,536 - 10

           = 65,526

262,144	131,072	65,536	32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Index register contents=131,073=   0 1 0   0 0 0 0   0 0 0 0   0 0 0 0   0 0 0 1

Partial address =   65,526=   1 1 1   1 1 1 1   1 1 1 1   1 1 1 1   0 1 1 0

                   Set to 1's by 16th bit

Effective address =   131,063=1   0 0 1   1 1 1 1   1 1 1 1   1 1 1 1   0 1 1 1

            This carry ignored.   Carry propagated between 16th and 17th bits.

- Incrementation

  When indexing in systems with the extended memory option, incrementation of
  the address contained in the index register is specified in the same manner
  as in systems not having this option; except in respect to the 16th bit of
  the partial address: Partial address = Displacement value.

<div align="center">NOTE</div>

  > The 16th bit of the partial address must be equal to 0.
  > See page 85.

  The 16th bit of the partial address being 0 has no effect on adder input
  bits 17, 18, and 19. A carry past the 19th bit cannot occur with the addi-
  tion of 65,536 as was the case in decrementation. This allows incrementa-
  tion from one memory group of 65,536 characters to the next higher address
  group.

  In the following example the index register is pointing to address 131,063
  which is located in the second group 65,536 characters. After incrementa-
  tion of 10 the effective address is in the third group of 65,536 characters.

  EXAMPLE:

  (Displacement value = 10)

  Partial address = Displacement value

  $$= 10$$

262,144	131,072	65,536	32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

  Index register contents=131,063= 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1

  Partial address =   10= 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0

  Effective address   131,073= 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

  Carry propagated between 16th and 17th bit.

- Compatibility

  In order to maintain compatibility with systems not having the extended
  memory option, the 16th bit of the partial address must be used (equal to
  1) when decrementing, but not when incrementing. When these conventions
  are observed, the displacement value assumes the limits - 32,768 to
  + 32,767.

Decrementation or negative displacement:

Partial address = 65,536 minus the displacement value

= 65,536 - 32,768 (maximum limit of displacement value)

= 32,768 (Lowest possible partial address with decremen-
tation. Only bit 16 is ON, giving the highest possible
displacement value.)

Incrementation or positive displacement:

Partial address = displacement value

= 32,767 (maximum displacement value and partial address
with bits 1-15=1 and bit 16=0)

It can be seen from the following examples that, although indexing in a system without the extended memory option may give the correct result, it is not necessarily compatible with systems having the extended memory option unless the limits of the displacement values have been adhered to.

EXAMPLE 1:

Assume that the address contained in an index register is 16,384 and the effective address required is 49,152.

INCREMENTATION

(Displacement value = 49,152 - 16,384)

= 32,768

Partial address = Displacement value

= 32,768

Without extended memory option -

32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Index register contents = 16,384 =

0 1 0 0　0 0 0 0　0 0 0 0　0 0 0 0

Partial address = 32,768 =

1 0 0 0　0 0 0 0　0 0 0 0　0 0 0 0

Effective address = 49,152 =

1 1 0 0　0 0 0 0　0 0 0 0　0 0 0 0

↑
Correct Result

With extended memory option -

262,144	131,072	65,536	32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Index register contents=16,384= 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Partial address =   32,768= 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

                              Set to 1's by 16th bit

Effective address = 507,904= 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

                       Incorrect Result

EXAMPLE 2:

Assume that the address contained in an index register is 33,792 and the effective address required is 992.

DECREMENTATION

(Displacement value = 33,792 - 992)
                    = 32,800

Partial Address = 65,536 minus the displacement value
              = 65,536 - 32,800
              = 32,736

Without extended memory option -

32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1
16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Index register contents = 33,792 = 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0

Partial address =   32,736 = 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0

Effective address =   992 = 1 0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0

       Correct Result   This carry ignored.

262,144	131,072	65,536	32,768	16,384	8,192	4,096	2,048	1,024	512	256	128	64	32	16	8	4	2	1
19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Index register contents=33,792= 0 0 0   1 0 0 0   0 1 0 0   0 0 0 0   0 0 0 0

Partial address =   32,736= 0 0 0   0 1 1 1   1 1 1 1   1 1 1 0   0 0 0 0

Effective address =   66,528= 0 0 1   0 0 0 0   0 0 1 1   1 1 1 0   0 0 0 0

         ↑   ↑

Incorrect Result   Carry propagated between 16th and 17th bits.

315 COMPATIBILITY OPTION

The 315 compatibility option permits the NCR Century 200 system to execute
315 programs through the use of a freestanding unit and three additional NCR
Century 200 commands. The unit contains the circuitry necessary to execute
the 315 commands, and includes a base address register (BAR), and a limit
address register (LAR). For a more detailed description refer to the 315
Compatibility option publication.

1401 COMPATIBILITY OPTION

The 1401 compatibility option permits the NCR Century 200 system to simulate
IBM 1240, 1401, 1440, and 1460 systems. The option consists of six commands
that work in conjunction with a freestand unit, the 627-202/203 emulation
unit. The emulation unit contains the circuitry necessary to set up IBM in-
structions. For more detailed discussions of this option, refer to the 1401
compatibility option publication.

FLOATING POINT OPTION

The floating point option for the NCR Century 200 System consists of the hard-
ware necessary to scale the numbers involved in a computation and to maintain
automatically the precision of the results of computation. Both single- and
double-precision floating point numbers may be used with the option, which
includes 12 additional hardware commands providing for addition, subtraction,
comparison, multiplication, multiplication-addition, and division. For a de-
tailed explanation of the floating point option, refer to the NCR Century 200
hardware commands and command timing publication in this series.
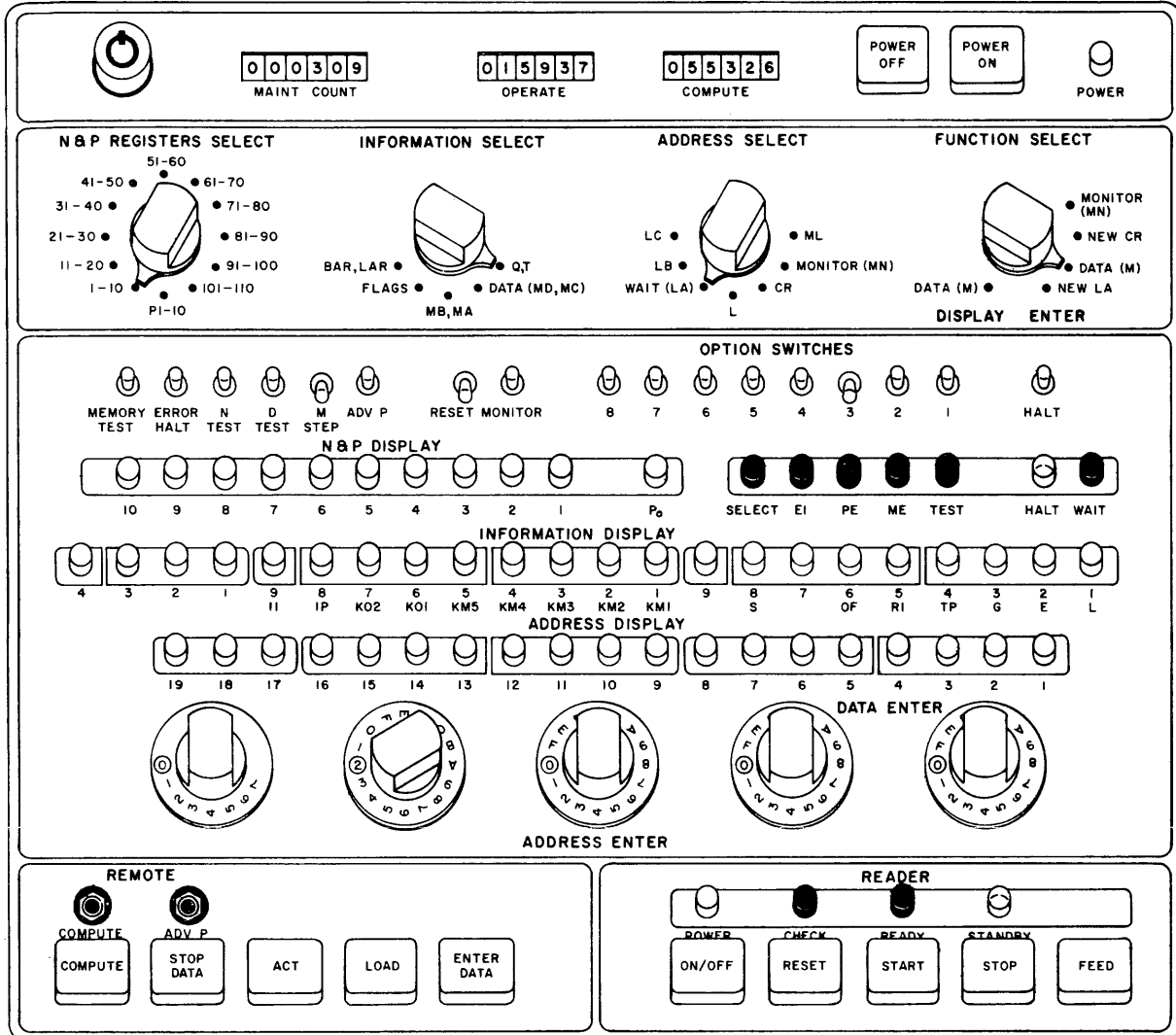
The operator's console, operating through position 1 of trunk 0, provides a means of communication between the operator and the computer. It permits the operator to display the contents of registers and memory locations, to alter memory contents, and to respond to system software and user program messages. For a detailed description of console operation, refer to the NCR Century OPERATORS INFORMATION MANUAL.

The console is equipped with an audible signal which alerts the operator that the system requires attention. The signal sounds when one of the following conditions occurs: the processor enters the wait state, the processor enters the error halt state, or a character with bit 8 ON is output to the I/O writer. The operator turns OFF the alarm either by setting the HALT switch ON or by pressing any character key on the I/O writer when the I/O writer is in the input or reset mode. The alarm can never be set ON when the HALT switch is ON or when the TEST switch is in the maintenance position.

INTRODUCTION

Every NCR Century 200 system is equipped with an integrated input/output (I/O) writer. The I/O writer provides a more convenient, faster method of operator-processor communications than the method provided by the use of the operator's console. The I/O writer permits keyboard entry of data, instructions, inquiries, and responses to the printed messages by software or user programs.

PHYSICAL DESCRIPTION

The I/O writer is a serial printing device with a keyboard. The standard I/O writer may be substituted with a thermal I/O writer.

Standard I/O Writer

The standard I/O writer is a serial printing device with a pin-feed platen that can accommodate three-part forms. The printing element is a rotating 64-character printhead. Character output by the keyboard is in standard ASCII code. Mechanical functions of the standard I/O writer include backspace, automatic line feed, automatic carriage return, and an audible end-of-line signal. Nominal data transfer rate of the standard I/O writer is six characters per second.

Thermal I/O Writer

The thermal I/O writer, which may be substituted for the standard I/O writer, is a serial, non-impact thermal printer. Printing is done by a printhead which consists of 35 elements, formed in a 7 x 5 - dot matrix. By heating the selected elements of the matrix and bringing them into light contact with the heat-sensitive paper, the character image is formed on the paper. Character output by the keyboard is in standard ASCII code. Mechanical functions of the thermal I/O writer include backspace, automatic line feed, automatic carriage return, and an audible end-of-line signal. The length of the printline is 80 characters and the data transfer rate is thirty characters per second.

Remote I/O Writer

Optionally, a second I/O writer may be added at a location not more than 500 feet from the console. The remote I/O writer operates in parallel with the console I/O writer, that is, characters output or input are printed simultaneously by both I/O writers. Once input is initiated on either system, the other unit may not initiate an input until the original input has terminated.

The I/O writer control does not maintain synchronization by monitoring data from the remote I/O writer when it is in the idle mode.

Both I/O writers on the system must be identical, that is, a standard I/O writer and a thermal I/O writer cannot be mixed on the system.

FUNCTIONAL DESCRIPTION

The functional description of the standard and thermal I/0 writers may be found in their respective product information publications, under the PRINTERS tab, in this manual.

PHYSICAL SPECIFICATIONS

PHYSICAL SPECIFICATIONS	
CHARACTERISTIC	SPECIFICATION
Power Requirements	3-phase 5-wire 6 AWG 120/208V 60HZ
KVA	12.0
BTU/HR	30,000
Current by leg	1 = 35.0 A 2 = 35.0 A 3 = 35.0 A
Circuit Breaker	3-pole 50 A
Convenience Outlet	15 A

For dimensions, weight, and access requirements of the individual components, see "Component Specifications," under SYSTEM INSTALLATION tab, in this manual.

ENVIRONMENTAL SPECIFICATIONS

The required environmental conditions for the NCR Century 200 Processor are as shown below.

OPERATING LIMITS	
Temperature	68° to 78°F Dry Bulb
Humidity	40% to 60% Relative
Altutude	7000 feet maximum

NCR CENTURY 200 HARDWARE COMMANDS AND COMMAND TIMING

INTRODUCTION

SCOPE AND PURPOSE OF THE PUBLICATION

This publication contains a functional description of the basic and optional hardware commands used by the NCR Century 200 System. It is intended as a supplement to the NCR Century 200 PROCESSOR manual, which is prerequisite reading, and is not a substitute for the NEAT/3 REFERENCE MANUAL. All command descriptions assume that:

1. A program written in NEAT/3 language has been compiled and is resident in memory in a form recognizable to processor hardware.
2. The command setup phase, described in detail in the NCR Century 200 PROCESSOR manual, has been completed.

Commands are described in terms of preserved values, both at the end of the setup phase and at the end of the execution phase.

HARDWARE COMMAND TIMING

Total command time is calculated as the sum of setup time (including indirect addressing and/or incremental indexing) plus command execution time.

$$C = S + E$$

Where:

C = Total time in nanoseconds required for setup and execution.
S = Total time in nanoseconds required for command setup.
E = Total time in nanoseconds required for command execution.

Command setup $S = (M_1A + 1)(PO + 2P) + RAP + M_2A(PO + 3P) + 2M_3AP +$
$$K[(M_1B + 1)(PO + 2P) + RBP + M_2B(PO + 3P) + 2M_3BP]$$

M_1A = Number of levels of mode 1 addressing for the A operand.

PO = 290 - 325 nanoseconds, average adder propagation time.

P = 1 memory cycle (760 nanoseconds average).

M_2A = 1 if mode 2 indirect addressing is used for the A operand; otherwise = 0

RA = Number of times indexing is required for the effective A address. This term will be at least 1 if mode 3 incremental indexing is specified.

M_3A = 1 if mode 3 incremental indexing is specified for the A operand; otherwise = 0.

K = 1 for the 2-address format of the command; K = 0 for the 1-address format.

M_1B = Number of levels of mode 1 indirect addressing for the B operand.

M_2B = 1 if mode 2 indirect addressing is specified for the B operand; otherwise = 0.

RB = Number of times indexing is required for the effective B address. This term will be at least 1 if mode 3 incremental indexing is specified.

M_3B = 1 if mode 3 incremental indexing is specified for the B operand; otherwise = 0.

M2A and M3A are mutually exclusive and may occur only once per command regardless of the level of mode 1 indirect addressing for the A operand. The same is true for M2B and M3B.

ADDRESS CONVENTION

The address of the leftmost character of every command must be equal to 0 modulo 4. Attempting to execute a command that violates this specification or to access a memory location greater than physical memory size results in a PE interruption.

COMMAND DESCRIPTIONS

The hardware-recognized commands for the NCR Century 200 System are tabulated and described in detail on the following pages. In addition to the 39 basic commands, 27 optional commands are also available with the system.

The codes in the following tables and in the command descriptions are given in three representations:

- The first representation is decimal; the binary equivalent of this number is the machine language command code in the 2-address version.
- The second representation (the first in parentheses) is the printout of the two-address code by the memory dump program.
- The third representation is the printout of the 1-address code by the memory dump program.

BASIC HARDWARE COMMANDS		
COMMAND	CODE	PAGE
ADD	64(40)(C0)	5
SUBTRACT	65(41)(C1)	10
MOVE B RIGHT TO LEFT	68(44)(C4)	15
COMPARE SIGNED	69(45)(C5)	17
SET IP ON	70(46)(C6)	21
SET IP OFF	71(47)(C7)	21
RESTORE	72(48)(C8)	22
EDIT	73(49)(C9)	23
COUNT	74(4A)(CA)	25
JUMP	75(4B)(CB)	25
PACK	76(4C)	
UNPACK	77(4D)(CD)	32
DECODE TO DELIMITER	78(4E)(CE)	37
DECODE ALL	79(4F)(CF)	40
OPTION SWITCHES INPUT	80(50)(D0)	43
TEST CHARACTER EQUAL	81(51)(D1)	43
TEST CHARACTER UNEQUAL	82(52)(D2)	44
TEST BIT	83(53)(D3)	44
MOVE A LEFT TO RIGHT	84(54)(D4)	45
SCAN ON KEY LESS THAN	85(55)(D5)	48
SCAN ON KEY EQUAL	86(56)(D6)	49
SCAN ON KEY GREATER THAN	87(57)(D7)	50
ADD BINARY	96(60)(E0)	52
SUBTRACT BINARY	97(61)(E1)	56
ADD UNSIGNED	98(62)(E2)	60
SUBTRACT UNSIGNED	99(63)(E3)	66
MOVE A RIGHT TO LEFT	100(64)(E4)	72
COMPARE BINARY	101(65)(E5)	75
REPEAT	102(66)(E6)	79
WAIT	103(67)(E7)	80
BRANCH OVERFLOW	104(68)(E8)	81
BRANCH LESS	105(69)(E9)	81
BRANCH EQUAL	106(6A)(EA)	81
BRANCH LESS OR EQUAL	107(6B)(EB)	81
BRANCH GREATER	108(6C)(EC)	81
BRANCH LESS OR GREATER	109(6D)(ED)	81
BRANCH GREATER OR EQUAL	110(6E)(EE)	81
BRANCH UNCONDITIONALLY	111(6F)(EF)	81
INOUT	112(70)(F0)	82

OPTIONAL HARDWARE COMMANDS		
COMMAND	CODE	PAGE
1401 ADD	56(38)(B8)	86
1401 SUBTRACT	57(39)(B9)	88
1401 SCAN	58(3A)(BA)	88
1401 MOVE	59(3B)(BB)	90
1401 CONVERT	60(3C)(BC)	92
315 EXECUTE	61(3D)(BD)	105
315 PACK (SPECIAL)	62(3E)(BE)	106
315 UNPACK (SPECIAL)	63(3F)(BF)	109
LOAD BAR	88(58)(D8)	111
LOAD MONITOR REGISTER	89(59)(D9)	112
LOAD TRACE	90(5A)(DA)	112
STORE TRACE	91(5B)(DB)	112
TABLE COMPARE	92(5C)(DC)	113
MULTIPLY	93(5D)(DD)	116
LOGIC	94(5E)(DE)	121
F.P. ADD SINGLE	116(74)(F4)	133
F.P. ADD DOUBLE	117(75)(F5)	154
F.P. SUBTRACT SINGLE	118(76)(F6)	164
F.P. SUBTRACT DOUBLE	119(77)(F7)	179
F.P. MULTIPLY & ADD SINGLE	120(78)(F8)	184
F.P. MULTIPLY & ADD DOUBLE	121(79)(F9)	185
F.P. COMPARE SINGLE	122(7A)(FA)	186
F.P. COMPARE DOUBLE	123(7B)(FB)	187
F.P. MULTIPLY SINGLE	124(7C)(FC)	188
F.P. MULTIPLY DOUBLE	125(7D)(FD)	193
F.P. DIVIDE SINGLE	126(7E)(FE)	197
F.P. DIVIDE DOUBLE	127(7F)(FF)	203

The following symbols and notations are used in the command descriptions:

A - The effective A address (after command setup).
B - The effective B address (after command setup).
T - The effective T value (after command setup).
() - The contents of, after command setup, and before execution;
    e.g., (B) = contents of B after command setup (the B operand).
X - The value X after command execution; e.g., T, (B).
C - The decimal equivalent of the binary value of the seven low-order
    bits of the command code character.

<u>NOTE</u>

T=T invariably, except for the RESTORE command. B is
specified for each command; however, the specification
is not necessarily correct if termination for a mal-
function occurs.

C = 64(40)(CO)

The contents of the field specified by the effective A address are added decimally to the field specified by the effective B address, and the results replace the contents of the B field. Each field is considered to contain signed, packed (see PACK command description) decimal information; the addition is algebraic. If the sign of (B) changes as a result of the addition, the configuration 1011 is stored for a positive result, and the configuration 1101, for a negative result.

If the absolute values of (A) and (B) are equal, the sign of (B) is unchanged by the operation, making it possible to obtain a negative field with the absolute value 0.

A carry beyond the leftmost character of (B) causes the overflow indicator (OI) to be set ON. This overflow carry does not affect memory, and, except for the carry, a true result is stored in the B field. If no overflow carry occurs, the OF is undisturbed.

T specifies the length of both (A) and (B) and ranges in value from 0 through 255, with 0 equivalent to 256.

If (A) and (B) overlap, the result may be different from the result obtained when (A) and (B) do not overlap.

Operation on nondecimal information yields a predictable result with no error indication.

$\underline{\bar{B}} = \bar{B}$

Command Execution Time

E = 2PO + 4P + 3T(P), if recomplementing is not necessary.
E = 3PO + 8P + 6T(P), if recomplementing is necessary.

EXAMPLE 1 (Like Signs)   All commands are represented in hexadecimal
                                    notation. Refer to the NCR Century 200 PROCESSOR
                                    manual for hexadecimal/decimal/binary conver-
                                    sion.

Q	RA	A2	A1	T	RB	B2	B1
40	50	01	3B	03	78	00	A0

Partial B address = 00160

IR 30
Mode 0

FL = 3

Partial A address = 00315

IR 20
Mode 0

CC for 2-address ADD command

Assume the index registers contain the following:

(IR20)   07D0 (02000)
(IR30)   0BB8 (03000)

After command setup:

Effective A address = 07D0 + 013B = 090B (02315)
Effective B address = 0BB8 + 00A0 = 0C58 (03160)

Before command execution:

	090B	090C	090D
(A)	0000 0111	0110 0101	0100 1011

Packed BCD value = 07654 +

sign

	0C58	0C59	0C5A
(B)	0000 0001	0000 0001	0001 1011

Packed BCD value = 01011 +

sign

Decimal equivalent of addition to be performed:

$$07654$$
$$01011$$
$$08665$$

1. Excess 6 is added to each A-field character as it is read from memory (see BCD arithmetic explanation in the NCR Century 200 processor publication):

   A-field characters   0000 0111 0110 0101 0100
   Excess 6   0110 0110 0110 0110 0110
   Partial sum   0110 0001 0000 0011 0010
   Carries   11   11   1   1
   Final sum   0110 1101 1100 1011 1010

2. B-field characters are added to the results obtained in step 1:

   Results of step 1   0110 1101 1100 1011 1010
   B-field characters   0000 0001 0000 0001 0001
   Partial sum   0110 1100 1100 1010 1011
   Carries   1   1
   Final sum   0110 1110 1100 1100 1011

3. Binary 6 is subtracted from the result of any character addition with a binary value >9 or no carry beyond the leftmost digit position:

   Final sums   0110 1110 1100 1100 1011
   Binary 6 correction 0110 0110 0110 0110 0110
   results   0000 1000 0110 0110 0101

4. The results of the addition replace the initial contents of the B field:

B	0C58	0C59	0C5A
(B)	0000 1000	0110 0110	0101 1011

   Packed BCD value = 08665 +

   sign

EXAMPLE 2 (Unlike Signs)

Assume that the fields used in the preceding example contained the following information:

A (A)	090B	090C	090D
	0000 0111	0110 0101	0100 1011

Packed BCD value = 7654 +

B (B)	0C58	0C59	0C5A
	0000 0001	0000 0001	0001 1101

Packed BCD value = 1011 –

Addition:

Decimal equivalent of addition to be performed:

$$7654+$$
$$1011-$$
$$\overline{6643+}$$

1. Because of unlike signs, the A characters are complemented before addition:

A characters	Complement
0000	1111
0111	1000
0110	1001
0101	1010
0100	1011

2. Each complemented A character is added to its corresponding B character:

```
complemented A character    1111 1000 1001 1010 1011
            B character     0000 0001 0000 0001 0001
                            ─────────────────────────
                            1111 1001 1001 1011 1100
```

3. Before storage in the B field, each character in the result is re-complemented:

Result of Addition	Complement
1111	0000
1001	0110
1001	0110
1011	0100
1100	0011

Following command execution:

__B__	0C58	0C59	0C5A
(B)	0000 0110	0110 0100	0011 1011

Packed BCD value = 6643 +

EXAMPLE 3 (Fields overlap)

Assume that the fields used in the preceding examples contained the following information and the B field began at memory address 090C:

A	090B	090C	090D
(A)	0000 0111	0110 0101	0100 1011

Packed BCD value = 07654 +

B	090C	090D	090E
(B)	0110 0101	0100 0011	0110 1011

Packed BCD value = 65436 +

Decimal equivalent of addition to be performed:

$$07654$$
$$\underline{65436}$$
$$73180$$

1. Excess 6 is added to each character of the A field.

A-field characters	0000 0111 0110 0101 0100
Excess 6	0110 0110 0110 0110 0110
Partial sum	0110 0001 0000 0011 0010
Carries	11 11 1 1
Final sum	0110 1101 1100 1011 1010

2. B-field characters are added to the result obtained in step 1.

Result of step 1	0110 1101 1100 1011 1010
B-field characters	0110 0101 0100 0011 0110
Partial sum	0000 1000 1000 1000 1100
Carries	11 1 1 1 1 11 1
Final sum (uncorrected)	1101 0011 0000 1110 0000

3. Binary 6 is subtracted from the result of any character addition with a binary value 9 or no carry beyond the leftmost bit position:

Results of addition	1101 0011 0001 1110 0000
Binary 6 correction	0110 0110
Final result	0111 0011 0001 1000 0000

Following command execution:

The A field is altered at locations 090C and 090D.

__B__	090C	090D	090E
(B)	0111 0011	0001 1000	0000 1011

Packed BCD equivalent = 73180 +

<center>Sign</center>

$$\underline{C \;=\; 65(41)(C1)}$$

(A) is decimally subtracted from (B), one character at a time, and the result replaces(B). Each field contains signed, packed, decimal information; the subtraction is algebraic. If the sign of (B) changes as a result of subtraction, the digit 1011 is stored for a positive result, and the digit 1101, for a negative result.

If the absolute values of (A) and (B) are equal, the sign of (B) is unchanged, making it possible to obtain a negative field with the absolute value zero.

T specifies the length of both (A) and (B) and ranges from 0 through 255, with 0 equivalent to 256.

A carry of the leftmost character of $\underline{(B)}$ causes the overflow indicator to be set ON. This overflow carry does not affect memory, and except for the carry, a true result is stored in B. If no overflow occurs, the overflow indicator is undisturbed.

If (A) and (B) overlap, the result may be different from the operation where (A) and (B) do not overlap.

Operation on nondecimal information gives a predictable result with no error indication.

$\underline{B} \;=\; B$

Command Execution Time

$E \;=\; 2PO + 4P + 3T(P)$, if recomplementing is not necessary.
$E \;=\; 3PO + 8P + 6T(P)$, if recomplementing is necessary.

EXAMPLE 1 (Like Signs)

Q	RA	A2	A1	T	RB	B2	B1
41	40	00	C8	04	B3	02	58

Partial B address = 00600

IR 44
Mode 3

FL = 4

Partial A address = 00200

IR 16
Mode 0

CC for 2-address SUBTRACT command
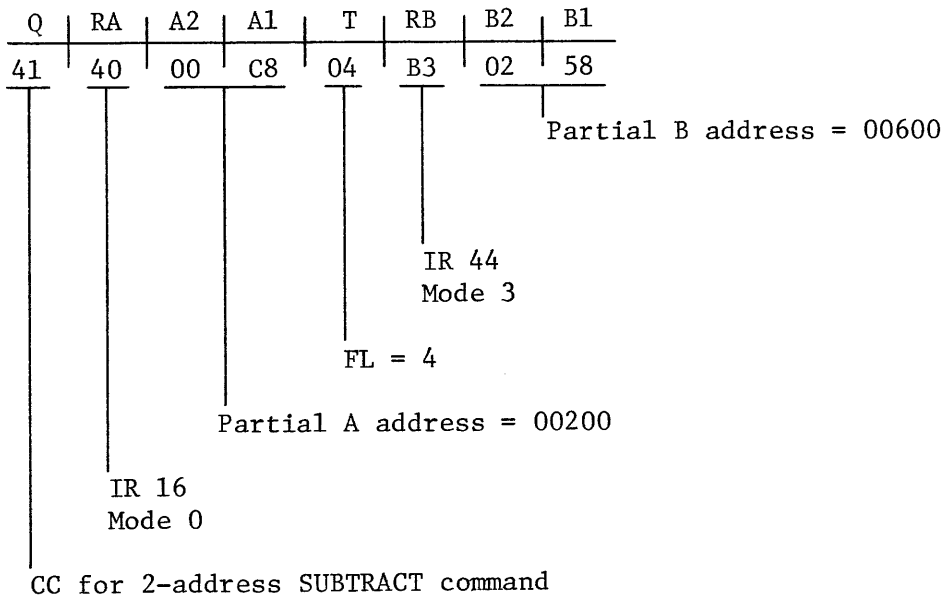
Assume the index registers contain the following:

(IR16) = 0640 (01600)
(IR44) = 1130 (04400)

After command setup:

Effective A address = 0640 + 00C8 = 0708 (01800)
Effective B address = 1130 + 0258 = 1388 (05000)

Before command execution:

A	0708	0709	070A	070B	
(A)	0100 0111	0011 0110	0010 1001	1000 1011	BCD 4736298 +

B	1388	1389	138A	138B	
(B)	0000 0111	0011 0110	0010 1001	1000 1011	BCD 0736298 +

sign

Since mode 3 was specified, (IR44) now equals 1388 (05000).

Decimal equivalent of subtraction to be performed:

$$
\begin{array}{rl}
\text{B} & 0736298 \\
\text{A} & \underline{4736298} \\
& -4000000
\end{array}
$$

1. Because of like signs, the A characters are complemented:

A characters	0100 0111 0011 1100 0010 1001 1000
Complements	1011 1000 1100 0011 1101 0110 0111

2. Each complemented A character is added to its corresponding B character:

Complemented A field	1011 1000 1100 0011 1101 0110 0111
B characters	0000 0111 0011 1100 0010 1001 1000
Results of addition	1011 1111 1111 1111 1111 1111 1111

3. Before storage in the B field each character of the result is re-complemented:

Result of addition =	1011 1111 1111 1111 1111 1111 1111
Recomplement =	0100 0000 0000 0000 0000 0000 0000

Following command execution:

The A-field is unchanged.

B	1388	1389	138A	138B
(B)	0100 0000	0000 0000	0000 0000	0000 1101

Packed BCD
equivalent = 4000000-

sign

EXAMPLE 2 (Unlike Signs).

Assume that the fields used in the preceding example contained the following:

A	0708	0709	070A	070B
(A)	0100 0111	0011 0110	0010 1001	1000 1011

Packed BCD
equivalent = 4736298+

B	1388	1389	138A	138B
(B)	0000 0111	0011 0110	0010 1001	1000 1101

Packed BCD
equivalent = 0736298-

Decimal equivalent of subtraction to be performed:

```
        SUBTRACT 0736298-              ADD 0736298-
                 4736298+      =           4736298-
                                          5472596-
```

1. Excess 6 is added to each character of the A field:

A-field characters	0100	0111	0011	0110	0010	1001	1000
Excess 6	0110	0110	0110	0110	0110	0110	0110
Partial sum	0010	0001	0101	0000	0100	1111	1110
Carries	1	11	1	11	1		
Final sum	1010	1101	1001	1100	1000	1111	1110

2. B-field characters are added to the result obtained in Step 1:

Results of Step 1	1010	1101	1001	1100	1000	1111	1110
B-field Characters	0000	0111	0011	0110	0010	1001	1000
Partial sum	1010	1010	1010	1010	1010	0110	0110
Carries		1 1	1	1		1	11
Final sum (uncorrected)	1011	0100	1101	0010	1011	1001	0110

3. Binary 6 is subtracted from the result of any character addition with
   a binary value > 9 or no carry beyond the leftmost bit position:

Results of addition	1011	0100	1101	0010	1011	1001	0110
Binary 6 correction	0110		0110		0110		
Final result	0101	0100	0111	0010	0101	1001	0110

Following command execution:

The A-field is unchanged.

B	1388	1389	138A	138B
(B)	0101 0100	0111 0010	0101 1001	0110 1101

Packed BCD equivalent =
5472596-

EXAMPLE 3 (fields overlap).

Assume that the fields used in the preceding example contained the following
information and the B field began at memory location 070A:

A	0708	0709	070A	070B
(A)	0000 0011	0100 0100	1000 1001	0011 1011

Packed BCD value =
0344893+

B	070A	070B	070C	070D
(B)	1000 1001	0011 1000	0011 0010	0001 1101

Packed BCD value =
8938321-

Decimal equivalent of subtraction (complementary addition) to be performed:

$$\text{SUBTRACT} \begin{array}{r} 8938321- \\ 0344893+ \end{array} \quad = \quad \text{ADD} \begin{array}{r} 8938321- \\ 0344893- \\ \hline 9283214- \end{array}$$

1. Excess 6 is added to each character of the A field:

A-field characters	0000	0011	0100	0100	1000	1001	0011
Excess 6		0110	0110	0110	0110	0110	0110	0110
Partial sum	0110	0101	0010	0010	1110	1111	0101
Carries		1	1	1			1
Final sum	0110	1001	1010	1010	1110	1111	1001

2. B-field characters are added to the results obtained in step 1:

Result of step 1	0110	1001	1010	1010	1110	1111	1001
B-field characters	1000	1001	0011	1000	0011	0010	0001
Partial sum	1110	0000	1001	0010	1101	1101	1000
Carries	1	1	1 1		1	1	1
Final sum (uncorrected)	1111	0010	1110	0011	0010	0001	1010

3. Binary 6 is subtracted from the result of any character addition with a binary value >9 or no carry beyond the leftmost bit position:

Result of addition	1111	0010	1110	0010	0010	0001	1010
Binary 6- correction	0110		0110				0110
Final result	1001	0010	1000	0011	0010	0001	0100

Following command execution:

A	0708	0709	070A	070B
(A)	0000 0011	0100 0100	1001 0010	1000 0011

Contents of overlap
area altered

B	070A	070B	070C	070C
(B)	1001 0010	1000 0011	0010 0001	0100 1101

Packed BCD value =
9283214-

Subtraction is not
affected by over-
lapping fields.

$$C = 68(44)(C4)$$

Each character of (B) is moved successively into the corresponding position in A, starting with the rightmost character. If (A) and (B) overlap, the results may be different from the results obtained when the fields do not overlap.

T specifies the number of characters of both (A) and (B) and ranges from 0 through 255 with zero equivalent to 256.

$\underline{B}$ = B

Command Execution Time

$E = 2PO + 2P + 2T(P)$

EXAMPLES

EXAMPLE 1

Assume that after command setup:

Effective A address = 0800 (02048)
Effective B address = 04B0 (01200)
T = 6

Before command execution:

A	0800	0801	0802	0803	0804	0805
(A)	0100 0110	0100 1001	0100 0101	0100 1100	0100 0100	0100 0001

ASCII = F   I   E   L   D   A

B	04B0	04B1	04B2	04B3	04B5	04B6
(B)	0100 0010	0100 0010	0100 0010	0100 0010	0100 0010	0100 0010

ASCII = B   B   B   B   B   B

Following command execution:

A	0800	0801	0802	0803	0804	0805
(A)	0100 0010	0100 0010	0100 0010	0100 0010	0100 0010	0100 0010

ASCII = B   B   B   B   B   B

The B field is unchanged.

EXAMPLE 2 (Fields Overlap)

Assume that the fields used in the previous example contain the following information and that the B field begins at address 0803.

A	0800	0801	0802	0803	0804	0805
(A)	0100 0110	0100 1001	0100 0101	0100 1100	0100 0100	0100 0001

| | F | I | E | L | D | A |

B	0803	0804	0805	0806	0807	0808
(B)	0100 1100	0100 0100	0100 0001	0101 1000	0101 1001	0101 1010

| | L | D | A | X | Y | Z |

When the move is initiated, the contents of 0808 replace the contents of 0805 in field A and the move continues as follows:

B	0806	0807	0808	0803	0804	0805
(B)	0101 1000	0101 1001	0101 1010	0101 1000	0101 1001	0101 1010

A	0803	0804	0805	0800	0801	0802
(A)	0101 1000	0101 1001	0101 1010	0101 1000	0101 1001	0101 1010
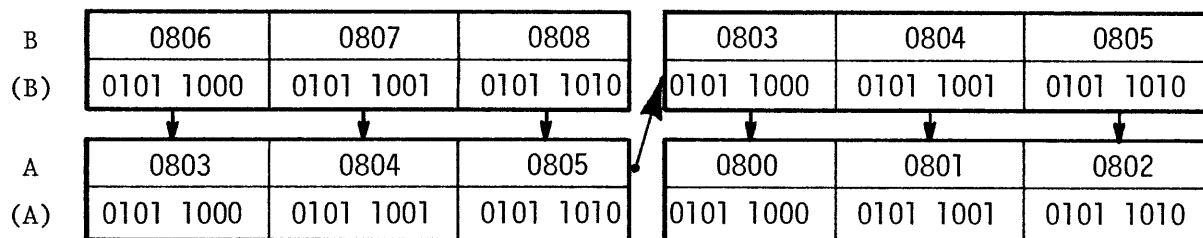
Following command execution:

A	0800	0801	0802	0803	0804	0805
(A)	0101 1000	0101 1001	0101 1010	0101 1000	0101 1001	0101 1010

| | X | Y | Z | X | Y | Z |

(B) = (A)

$$\underline{C \ = \ 69(45)(C5)}$$

(A) is compared to (B). Both operands are considered to be signed, packed fields.

One of the LEG flags is set ON, indicating that (A) is less than, equal to, or greater than (B); the other two flags are set OFF.

When two fields having zero in all positions except the sign position are compared, the result is equal regardless of signs.

T specifies the length of (A) and (B) and ranges from 0 through 255 with zero equivalent to 256.

Operation on nondecimal information gives a predictable result with no error indication.

$\underline{B} \ = \ B$

Command Execution Time
<u>Command Execution Time</u>

$E \ = \ 2PO \ + \ 4P \ + \ 3T(P)$

EXAMPLE 1 (A) > (B)

```
  Q | RA | A2 | A1 | T | RB | B2 | B1
 45 | 00 | 08 | 00 | 03| 30 | 00 | 00
```

Partial B address = 00000

IR 12
Mode 0

FL = 3

Partial A address = 02048

IR (None)
Mode 0
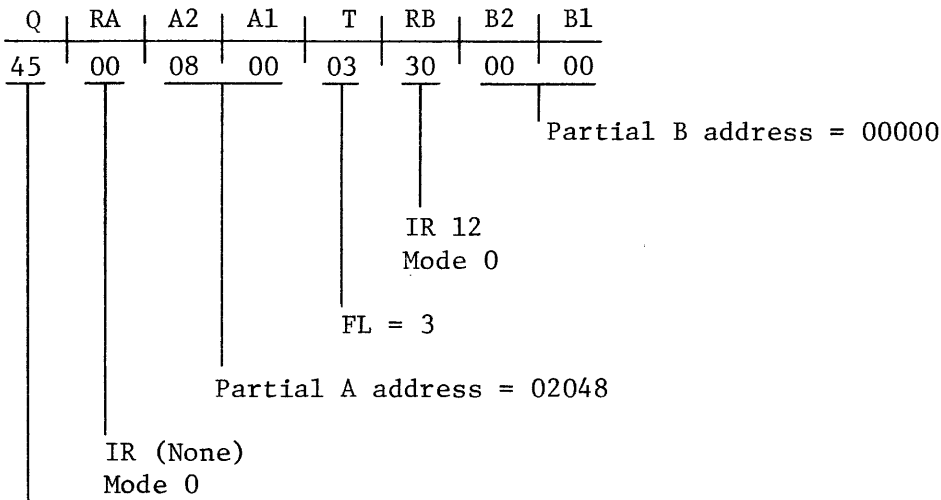
CC for 2-address COMPARE SIGNED command

Assume the index register contains the following:

(IR12) = 04B0 (01200)

After command setup:

Effective A address = 0800 (02048)
Effective B address = 04B0 + 00000 = 04B0 (01200)

Before command execution:

A	0800	0801	0802
(A)	0010 0100	0110 1000	0101 1001

Packed BCD value = 24685+

B	04B0	04B1	04B2
(B)	0001 0011	0110 1001	1000 1011

Packed BCD value = 13698+

Since the A and B fields have like signs, the A field is compared to the
B field as follows:

1. The 1's complement is derived for each A-field character except the sign:

    A-field characters   0010 0100   complements   1101 1011
                          0110 1000   1001 0111
                          0101   1010

2. The 1's complements of the A-field characters are added to the corresponding B-field characters:

    A-field complements   1101 1011 1001 0111 1010
    B-field characters   0001 0011 0110 1001 1000
    Partial Sum   1100 1000 1111 1110 0010
    Carries   1   11   11
    Initial carry   1
    Final Sum   1110 1111 0000 0001 0011

Following command execution:

1. The A and B fields are unchanged.
2. The number unequal indicator is turned ON (final sum ≠ 0).
3. Since there was no carry beyond the specified field length and the A-field is positive, the G flag is turned ON.

EXAMPLE 2 (A) < (B)

Assume that the A and B fields from the previous example are used and that the A field is now negative. Memory location 0802 now contains 0101 1101.

1. The signs of the A and B fields are compared.
2. The 1's complements of the A-field characters are added to the corresponding B-field characters exactly as shown in the previous example, with the same result.

Following command execution:

1. The A and B fields are unchanged.
2. Since the A field is negative and a carry beyond the leftmost bit position of the specified length does not exist, the L flag is turned ON.

EXAMPLE 3 (A) = (B)

Assume that the fields defined in example 1 had contained the following information:

A	0800	0801	0802
(A)	0100 0001	0011 1000	0110 1101

Packed BCD value = 41386–

B	04B0	04B1	04B2
(B)	0100 0001	0011 1000	0110 1101

Packed BCD value = 41386–

1. The signs of the A and B fields are compared.
2. The A and B fields are compared:

1's complement of (A)	1011 1110 1100 1000 1001
Characters of (B)	0100 0001 0011 1000 0110
Partial sum	1111 1111 1111 1111 1111
Initial carry	1
Final sum	1 0000 0000 0000 0000 0000

Following command execution:

1. The A and B fields are unchanged.
2. Since the signs of the A and B fields are alike and the result is all zeros, the E flag is turned ON.

NOTE

In the above example the result is zero; however, if the signs of the A and B fields were unlike, the G or L flag would be turned ON instead of the E flag.

C = 70(46)(C6)

The interrupt permit (IP) is turned ON. The repeat indicator (RI) is turned OFF. Program control is transferred to A. If A is not a legal command address, a PE occurs and the control register is undisturbed.

When this command terminates, the tests for trace trapping and interrupt trapping are bypassed.

T and B are not used.

C = 71(47)(C7)

The interrupt permit and repeat indicators are turned OFF. Program control is transferred to A. If A is not a legal command address, a PE occurs and the control register is undisturbed.

T and B are not used.

$$\underline{C = 72(48)(C8)}$$

(A) is an 8-character field, called a status word, which sets up certain values that establish or reestablish a desired program state.

A must be equal to 0 modulo 4 or a PE results.

Correspondence between memory locations and the values affected is as follows:

LOCATION	VALUES
A	T
A + 1, 2, AND 3	LOW ORDER 16 OR 19 BITS OF B OPERAND (19 BITS IF EXTENDED MEMORY).
A + 4	FLAGS AND INDICATORS, AS FOLLOWS:
B8	USED FOR MULTIPROGRAMMING. OTHERWISE "0"
B7	NOT SPECIFIED AND WILL BE 0.
B6	OF FLAG
B5	REPEAT INDICATOR
B4	TRACE PERMIT
B3	G FLAG
B2	E FLAG
B1	L FLAG
A + 5, 6, AND 7	LOW ORDER 16 OR 19 BITS OF CONTROL REGISTER (19 BITS IF EXTENDED MEMORY).

After the B and T values are established, the contents of A + 5, 6, and 7 are checked to ensure that a legal command address would be loaded into the control register; a violation produces an immediate PE, with the control register undisturbed.

The RESTORE command bypasses testing the RI and the trace permit indicator (TP).

T and B are not used but reflect the contents of A and A + 1, 2, 3.

Command Execution Time

E = PO + 5P

$$C = 73(49)(C9)$$

(B) contains a pattern into which (A) is edited.

(A) is moved to B sequentially, one character at a time, according to the pattern stored in B. Starting with the leftmost character of each field, the first character of (A) is moved into the first available position of B. A position is available if the character occupying it has a $b7 = 1$; otherwise (if $b7 = 0$), the position is not disturbed.

When the rightmost character of (A) is an ASCII "plus", an ASCII "space" is stored in the rightmost position of B.

(B) contains (A) interspersed with those characters of (B) where $b7 = 0$.

$\underline{B} = B + T + N$, where N is the number of characters with $b7 = 0$; if $T = 0$, then $\underline{B} = B + 256 + N$.

Command Execution Time

$E = PO + 3 (T + n) P$ where n is the number of B operand characters with $b7 = 0$ that are skipped over.

Following command setup assume that:

Effective A address = 0898 (02200)
Effective B address = 0BB8 (03000)

T = 5

Before command execution:

A	0898	0899	089A	089B	089C
(A)	0011 0010	0011 0010	0011 0011	0011 0111	0010 1011
ASCII	2	2	3	7	+

B	0BB8	0BB9	0BBA	0BBB	0BBC
(B)	0010 0100	0010 1010	0101 1000	0101 1000	0010 1110
ASCII	$	*	X	X	.

Following command execution:

B	0BB8	0BB9	0BBA	0BBB	0BBC	0BBD	0BBE	0BBF
(B)	0010 0100	0010 1010	0011 0010	0011 0010	0010 1110	0011 0011	0011 0111	0010 0000
ASCII	$	*	2	2	.	3	7	sp

<u>NOTE</u>

The length of B is increased by the number of bytes
containing a configuration where $b7 = 0$. Since T = 5
and the number of undisturbed bytes is three (0BB8,
0BB9 and 0BBC) in the B field, B becomes 8 bytes in
length.

C = 74(4A)(CA)

The 1-character binary field at memory location 64 (called the COUNT counter) is accessed, decremented by 1, and restored to memory. The resulting value is then tested equal to 0 (0 decremented by 1 = 255). If the result is not equal to 0, program control is transferred to A. If the result is equal to 0, the next command in sequence is executed. In either case, the repeat indicator is turned OFF.

B and T are not used.

Command Execution Time:

E = PO + 2P if the COUNT counter is decremented to zero.
E = PO + 3P if the COUNT counter is not decremented to zero.

C = 75(4B)(CB)

Program control is unconditionally transferred to A; the repeat indicator (RI) and the error indicator (EI) are turned OFF. If A is not a legal command address, a PE occurs and the control register is undisturbed.

The address of the character following the JUMP command (the link address) is stored in the jump link register (IR8).

B and T are not used.

Command Execution Time

E = PO + 3P

$$C = 76(4C)(CC)$$

The contents of the field specified by the effective A address are packed into the field specified by the effective B address. That is, the least significant four bits (b4-b1) of each character in a pair of A characters are combined to form a single 8-bit character which is stored in one B character.

Packing is performed sequentially, from left to right, starting at the low-order A and B addresses. The four least significant bits of the left character of the A pair replace the four most significant bits of the B character; the four least significant bits of the right character of the A pair replace the four least significant bits of the B character:

```
           A           A + 1         A + 2         A + 3
    ┌───────────┬───────────┬───────────┬───────────┐
    │ 0011 0110 │ 0011 0010 │ 0011 0100 │ 0010 1101 │
    └───────────┴───────────┴───────────┴───────────┘

              ┌───────────┬───────────┐
              │ 0110 0010 │ 0100 1101 │
              └───────────┴───────────┘
                   B           B + 1
```
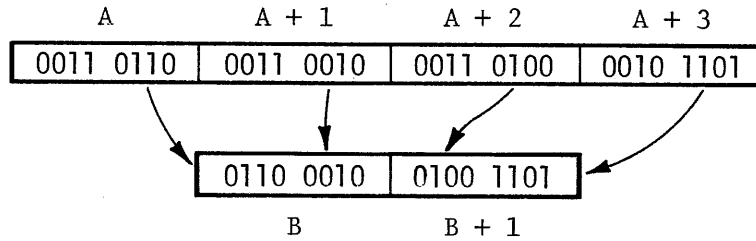
Packing should be performed only on those characters shown in the shaded area on the following chart:

NCR CENTURY CODE CHART

B8-B5 \ B4-B1		0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0000	0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
0001	1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
0010	2	SP	!	"	#	$	%	&	/	(	)	*	+	,	-	.	/
0011	3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0100	4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0101	5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	'	—
0110	6	\	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0111	7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

These are the only characters that can be restored to their original values by the UNPACK command, as explained on page 32. If any other characters are packed, they lose their original values.

The T portion of the command specifies the number of characters to be packed (length of the A field) and may be any value from 0 through 255, which 0 considered to be 256. The number of B characters affected is T/2 if T is even, and (T + 1)/2 if T is odd. Where T is odd, the leftmost character of the first A pair is treated as the odd character to be right-justified in the left character of the B field; that is, b4-b1 of the A field character are moved to b4-b1 of the B field character, with b8-b5 of the B field character set to zeros.

T odd:

```
         A            A + 1          A + 2
    ┌───────────┬───────────┬───────────┐
    │ 0011 0010 │ 0011 0100 │ 0010 1101 │
    └───────────┴───────────┴───────────┘
             ↘          ↓          ↙
       ┌───────────┬───────────┐
       │ 0000 0010 │ 0100 1101 │
       └───────────┴───────────┘
             B           B + 1
```
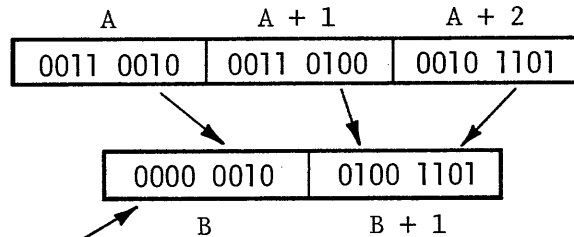
Zeros stored in four most-significant bits of left-most B character.

When the command is completed, the field that was designated by the effective A address retains its initial contents; the field that was designated by the effective B address contains the packed data from the A field.

If the A and B fields overlap, the final results may be different from the results of an operation in which the fields do not overlap. In this case, the initial contents of the A field are altered.

$\underline{B}$ = B + T/2 if T is even and not equal to zero.
$\underline{B}$ = B + (T + 1)/2 if T is odd.
$\underline{B}$ = T + 128 if T = 0.

Command Execution Time

$$E = PO + \left(\frac{3T}{2}\right) P, \text{ if T is even}$$

$$E = PO + \left[\frac{3(T+1)}{2}\right] P, \text{ if T is odd}$$

EXAMPLE 1
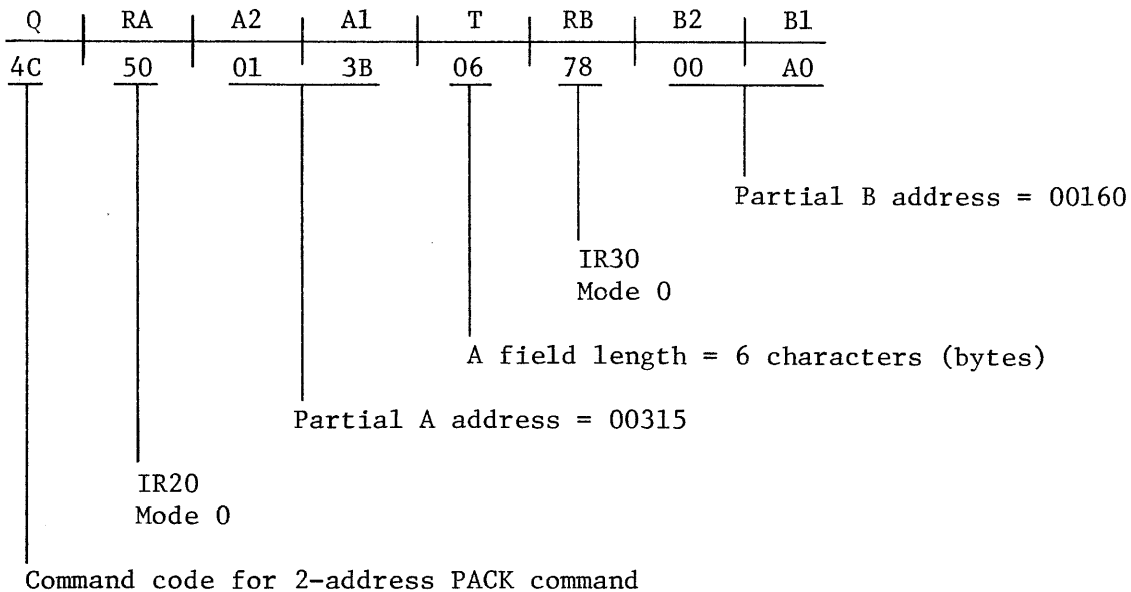
Q	RA	A2	A1	T	RB	B2	B1
4C	50	01	3B	06	78	00	A0

Partial B address = 00160

IR30
Mode 0

A field length = 6 characters (bytes)

Partial A address = 00315

IR20
Mode 0

Command code for 2-address PACK command

Assume the index registers contain the following:

(IR20) = 07D0 (02000)
(IR30) = 0BB8 (03000)

After command setup:

Effective A address = 07D0 + 013B = 090B (02315)
Effective B address = 0BB8 + 00A0 = 0C58 (03160)

Before command execution, the A field contains the following information
(the contents of the B field are not significant):

A	090B	090C	090D	090E	090F	0910
(A)	0011 0000	0011 0011	0011 0010	0011 0100	0011 1001	0010 1101

Following command execution, the contents of the A field are unchanged, the
contents of the B field are:

B	0C58	0C59	0C5A
(B)	0000 0011	0010 0100	1001 1101

B = 0C58 + 03 = 0C5B(03163)

EXAMPLE 2 (T is odd)

Q	RA	A2	A1	T	RB	B2	B1
4C	64	01	F4	07	40	00	C8

Partial B address (00200)

IR16
Mode 0

FL = 7

Partial A address (00500)

IR25
Mode 0

CC for 2-address PACK command
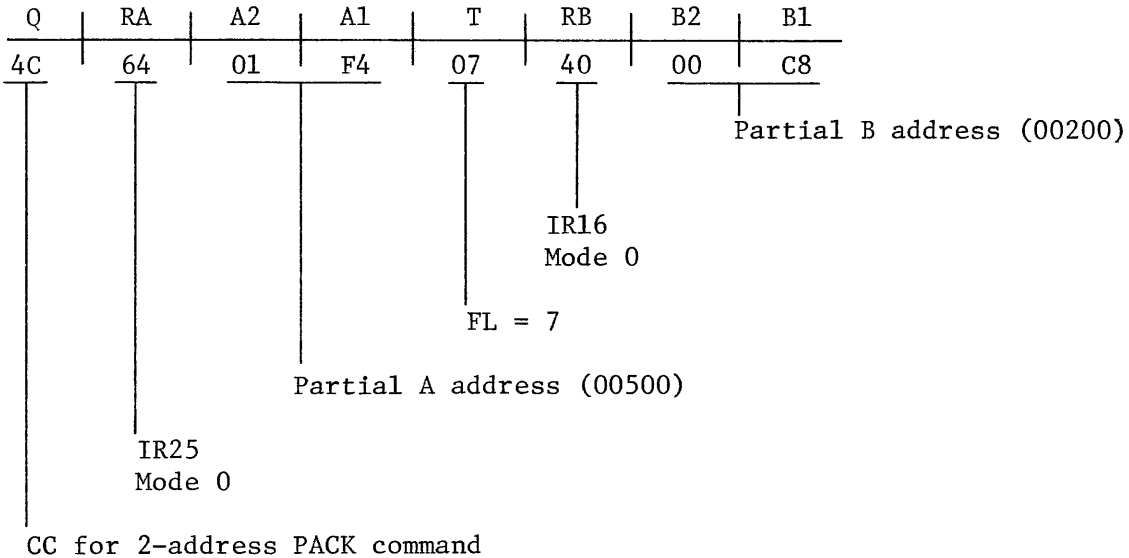
Assume the index registers contain the following:

(IR16) = 0640 (01600)
(IR25) = 09C4 (02500)

After command setup:

Effective A address = 09C4 + 01F4 = 0BB8 (03000)
Effective B address = 0640 + 00C8 = 0708 (01800)

Before command execution:

A	0BB8	0BB9	0BBA	0BBB	0BBC	0BBD	0BBE
(A)	0010 1010	0010 0010	0011 0010	0011 0011	0011 0110	0011 0101	0010 1011

After command execution:

B	0708	0709	070A	070B
(B)	0000 1010	0010 0010	0011 0110	0101 1011

B = 0708 + 04 = 070C (01804)

EXAMPLE 3 (Fields Overlap)

```
 Q  |  RA  |  A2  |  A1  |  T  |  RB  |  B2  |  B1
4C  |  28  |  00     63  | 06  |  00  |  04  |  4D
```

Partial B address (01101)

No indexing

FL = 6

Partial A address (00099)

IR10
Mode 0

CC for 2-address PACK command
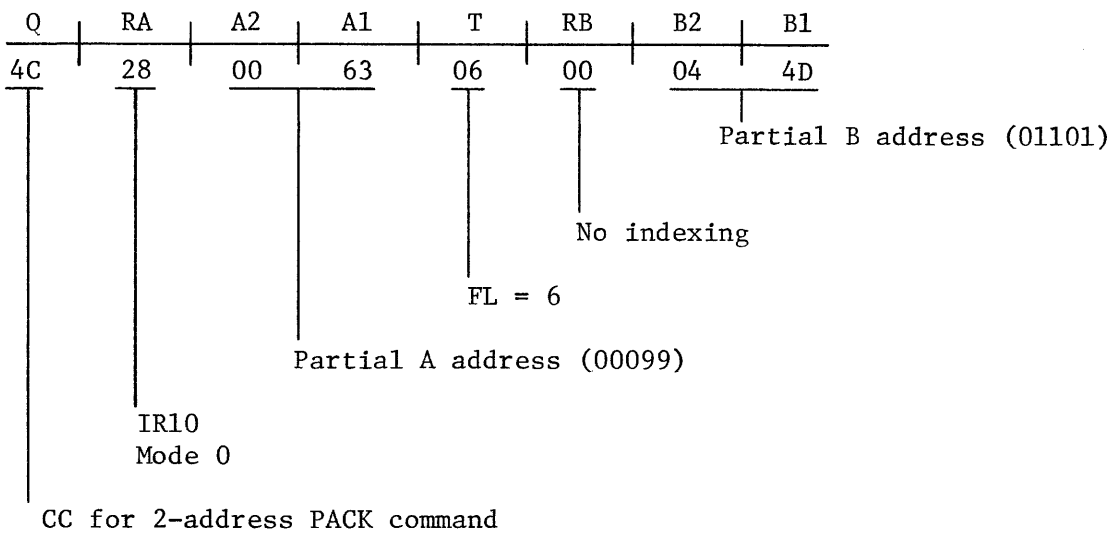
Assume IR10 contains 03E8 (01000)

After command setup:

Effective A address = 03E8 + 0063 = 044B (01099)
Effective B address = 044D (01101) since no indexing is required.

Before command execution:

		This portion of the A field overlaps the B field.			
A	044B	044C	044D	044E	044F	0450
(A)	0011 0001	0011 0111	0011 1001	0011 0111	0011 0110	0011 0101

Because the fields overlap, they are altered during command execution as
follows:

1. B4 - b1 of 044B replace b8 - b5 of 044D:

044B	044D
0011 0001	0001 1001
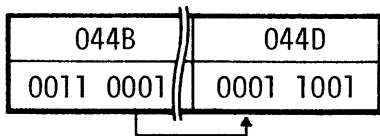
2. B4 - b1 of 044C replace b4 - b1 of 044D:

044C	044D
0011 0111	0001 0111
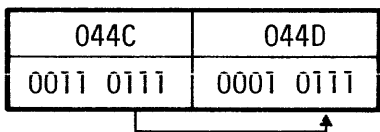
3. B4 - b1 of 044D, which were altered in step 1, replace b8 - b5 of 044E:

044D	044E
0001 0111	0111 0111

4. B4 - b1 of 044E remain unchanged, since, in effect, they replace themselves.

5. B4 - b1 of 044F replace b8 - b5 of 044F:

044F
0110 0110

6. B4 - b1 of 0450 replace b4 - b1 of 044F:

044F	0450
0110 0101	0011 0101

7. Following command execution:

A (A)	044B	044C	044D	044E	044F	0450
	0011 0001	0011 0111	0001 0111	0111 0111	0110 0101	0011 0101

B (B)	044D	044E	044F
	0001 0111	0111 0111	0110 0101

B = 044D + 03 = 0450 (01104).

C = 77(4D)(CD)

The contents of the field specified by the effective A address are unpacked into the field specified by the effective B address. That is, each 8-bit A character is divided into two 4-bit digits. Appropriate zone bits are appended to each 4-bit character forming two NCR Century ASCII characters. The two ASCII characters are then stored in a pair of adjacent locations in the B field.

Unpacking is performed sequentially, from left to right, starting at the low-order A and B addresses. Bits b8 - b5 of each A character replace b4 - b1 of the left B character in each pair, and the appended zone bits are stored in b8 - b5 of the B characters. Bits b4 - b1 of each A character replace b4 - b1 of the right B character in each pair, the zone bits again occupying b8 - b5.

If the binary value of the 4-bit A character is nine or less, the zone bits 0011 are appended; if the binary value is greater than nine, the zone bits 0010 are appended:

```
            A           A + 1
       ┌──────────┬──────────┐
       │ 0110 0010│ 0100 1101│
       └──────────┴──────────┘
          ╲   ╲       ╲       ╲
           ╲   ╲       ╲       ╲
   ┌──────────┬──────────┬──────────┬──────────┐
   │ 0011 0110│ 0011 0010│ 0011 0100│ 0010 1101│
   └──────────┴──────────┴──────────┴──────────┘
        B         B + 1       B + 2      B + 3
```
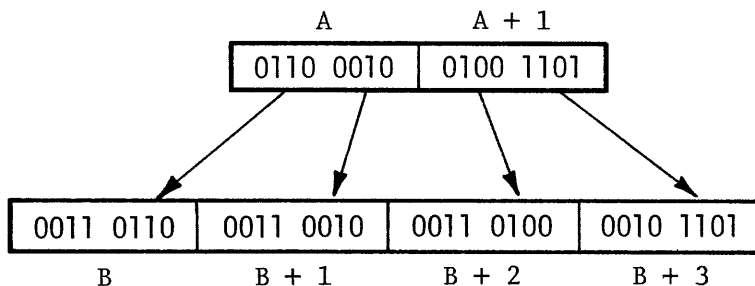
The T portion of the command specifies the number of characters to be unpacked (length of the field designated by the effective A operand). T may range in value from 0 through 255, with 0 considered equal to 256. The number of B characters affected is either 2T (T $\neq$ 0) or 512 (T=0).

When command execution is complete, the A field retains its initial contents; the B field's initial contents are replaced by the unpacked data.

If the A and B fields overlap, the results may be different from the results obtained when the two fields do not overlap.

$\underline{B}$ = B + 2T if T is not equal to 0.

$\underline{B}$ = B + 512 if T is equal to 0.

Command Execution Time
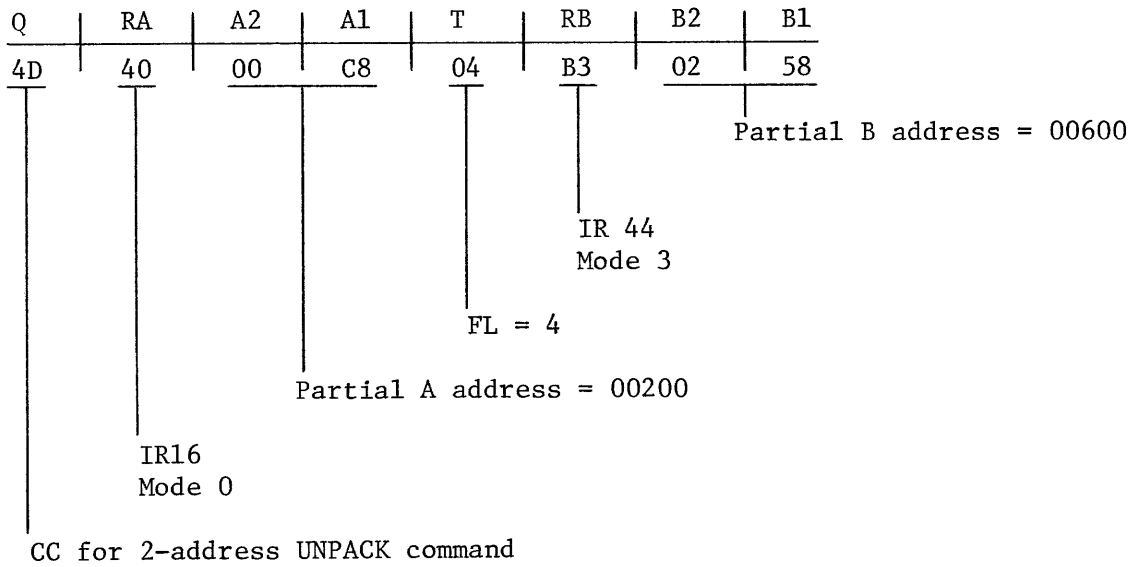
E = PO + 3T(P)

EXAMPLE 1

Q	RA	A2	A1	T	RB	B2	B1
4D	40	00	C8	04	B3	02	58

Partial B address = 00600

IR 44
Mode 3

FL = 4

Partial A address = 00200

IR16
Mode 0

CC for 2-address UNPACK command

Assume the index registers contain the following:

(IR16) = 0640 (01600)
(IR44) = 1130 (04400)

After command setup:

Effective A address = 0640 + 00C8 = 0708 (01800)
Effective B address = 1130 + 0258 = 1388 (05000)
(IR44) = 1388 (05000) since mode 3 (incremental indexing) was specified.
Before command execution:

A	0708	0709	070A	070B
(A)	0000 1010	1010 0010	0101 0110	0011 1011

The contents of B after command setup are not significant.

Following command execution:

B	1388	1389	138A	138B	138C
(B)	0011 0000	0010 1010	0010 1010	0011 0010	0011 0101

138D	138E	138F
0011 0110	0011 0011	0010 1011

(A) is unchanged.

B = 1388 + 8 = 1390 (05008)

EXAMPLE 2 (Fields overlap)

Q	RA	A2	A1	T	RB	B2	B1
4D	34	01	A6	04	00	34	6D

Partial B address = 13421

No IR

FL = 4

Partial A address = 00422
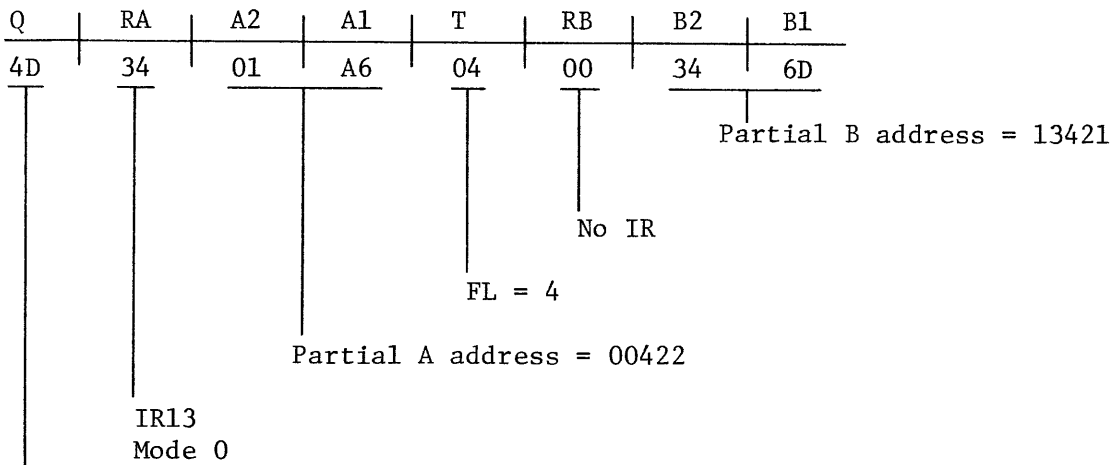
IR13
Mode 0

CC for 2-address UNPACK command

Assume the index register contains the following:

(IR13) = 32C8 (13000)

After command setup:

Effective A address = 32C8 + 01A6 = 346E (13422)
Effective B address = 346D (13421) - no indexing required

Before command execution:

	OVERLAPPING AREAS			
346D	346E	346F	3470	3471
0000 0000	0000 0010	0111 0100	0011 0110	0100 1101

3472	3473	3474
0000 0000	0000 0000	0000 0000

Because the fields overlap, they are altered during command execution as follows:

1. B8 - b5 of 346E replace b4 - b1 of 346D; zone bits 0011 are appended:

346D	346E
0011 0000	0000 0010
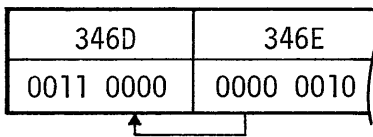
2. B4 - b1 346E remain unchanged, since, in effect, they replace themselves. Zone bits 0011 replace b8 - b5 of 346E:

346E	346E
0000 0010	0011 0010
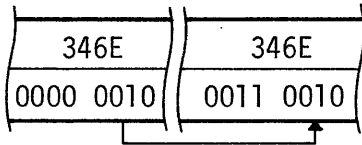
3. B8 - b5 of 346F replace b4 - b1 of 346F. Zone bits 0011 replace b8 - b5 of 346F:

346F	346F
0111 0100	0011 0111
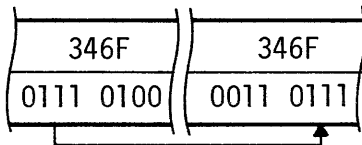
4. B4 - b1 of 346F replace b4 - b1 of 3470. When the contents of 346F were originally setup for unpacking, all eight bits were moved to a temporary storage area and unpacked from there so that the original b4 - b1 bits of 346F (0100), not the bits obtained in step 3 (0111), replace b4 - b1 of 3470:

346F	3470
0111 0100	0011 0100
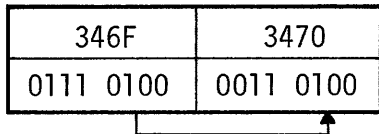
5. B8 - b5 of 3470, altered in step 4, replace b4 - b1 of 3471:

3470	3471
0011 0100	0011 0011
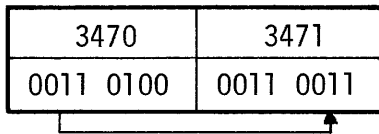
6. B4 - b1 of 3470, altered in step 4, replace b4 - b1 of 3472:

3470	3472
0011 0100	0011 0100
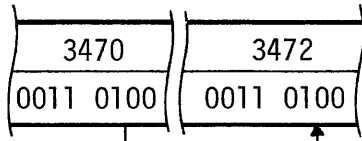
7. B8 - b5 of 3471, altered in Step 5, replace b4 - b1 of 3473:

3471	3473
0011 0011	0011 0011
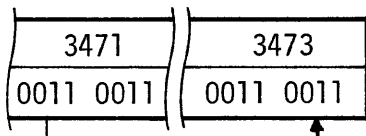
8. B4 – b1 of 3471, altered in step 6, replace b4 – b1 of 3474:

3471	3474
0011 0011	0011 0011
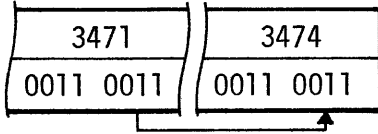
Following command execution:

B	346D	346E	346F	3470	3471	3472	3473	3474
(B)	0011 0000	0011 0010	0011 0111	0011 0100	0011 0011	0011 0100	0011 0011	0011 0011

A	346E	346F	3470	3471
(A)	0011 0010	0011 0111	0011 0100	0011 0011

$\underline{B}$ = 346D + 08 = 3475 (13429)

C = 78(4E)(CE)

(A) designates the beginning address of a table of replacement characters for the B field. To compute the replacement character for (B), the eight bits of a B character are added to the eight least-significant bits of the A address. The resulting memory address is read out and the replacement character is stored in the corresponding B position from left to right.

The command terminates when a replacement character has a 1 bit in b8 or (B) is exhausted, whichever comes first. If the command terminates because the b8 delimiter is encountered, the character is not stored into (B) and the E flag is turned ON. If the B field is exhausted before a character with b8 on is encountered in the A table, the G flag is turned ON.

The address + 1 of the last character of (B) to be replaced is stored in the next address index register (IR9).

T specifies the length of (B) and ranges from 0 through 255, with 0 equivalent to 256.

B = (IR9)

Command Execution Time

E = 2PO + 6P + 4N(P)

Where:

N = The number of nondelimiter characters decoded in the field;
$0 < N < (T-1)$

Q	RA	A2	A1	T	RB	B2	B1
4E	B3	02	58	05	40	00	C8

Partial B address = 00200

IR16
Mode 0

FL = 5

Partial A address = 00600
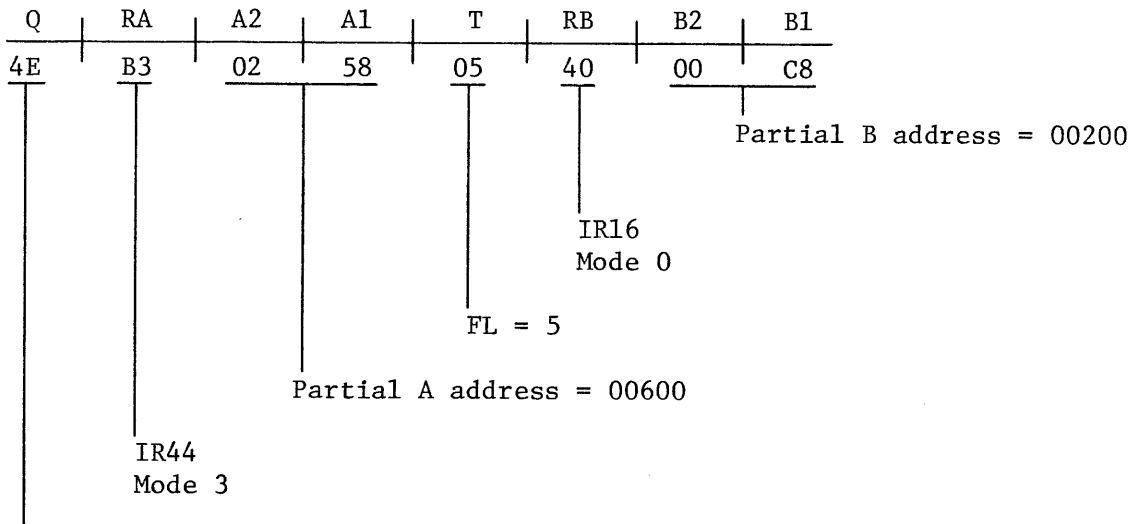
IR44
Mode 3

CC for 2-address DECODE TO DELIMITER command

Assume the index registers contain the following:

(IR44) = 1130 (04400)
(IR16) = 0640 (01600)

After command setup:

Effective A address = 1130 + 0258 = 1388 (05000)
Effective B address = 0640 + 00C8 = 0708 (01800)

Before command execution:

A	1388	1389
(A)	0001 1100	0111 1010

BCD = 1C7A (07290)

B	0708	0709	070A	070B	070C
(B)	0011 0011	0000 1100	0101 0011	0100 1100	0110 0000
BCD =	33	0C	53	4C	50

<u>NOTE</u>

Since mode 3 was specified, (IR44) equals 1388 (05000).

Following command execution:

B	0708	0709	070A	070B	070C
(B)	0111 0111	0001 0010	0101 0011	0100 1100	0110 0000

Replacement characters not
stored in B; delimiter
found in memory location
ICDD.

Replacement character
from memory location
1C86.

Replacement character
from memory location
1CAD.

Replacement character formation:

1CAD
0111 0111

ADD   1C7A   contents of A (1388-89)
        33   contents of B (0708)
    1CAD   replacement characters address

1C86
0001 0010

ADD   1C7A   contents of A
        0C   contents of B plus 1- (0709)
    1C86   replacement characters address

1CDD
1111 0000

ADD   1C7A   contents of A
        53   contents of B plus 2- (070A)
    1CDD   replacement characters address

Delimiter

The A field remains unchanged.
(IR9) equals 070A.
The E flag is ON.

C = 79(4F)(CF)

The DECODE ALL command is similar to the DECODE TO DELIMITER command. The only difference between the two operations is that DECODE ALL causes the entire B field to be decoded. A 1 bit in b8 of a replacement character will not cause command termination.

The address B + T is stored in the next address index register (IR9), unless T = 0, in which case B + 256 is stored in the NAIR. The G flag is turned ON.

T specifies the length of (B) and ranges from 0 through 255, with 0 equivalent to 256.

<u>B</u> = (IR9)

Command Execution Time

E = 2PO + 2P + 4T(P)

```
  Q  |  RA  |  A2  |  A1  |  T  |  RB  |  B2  |  B1
  4F |  B3  |  02  |  58  |  04 |  40  |  00  |  C8
```

Partial B address = 00200

IR 16
Mode 0

FL = 04

Partial A address = 00600

IR 44
Mode 3

CC for 2-address DECODE ALL command
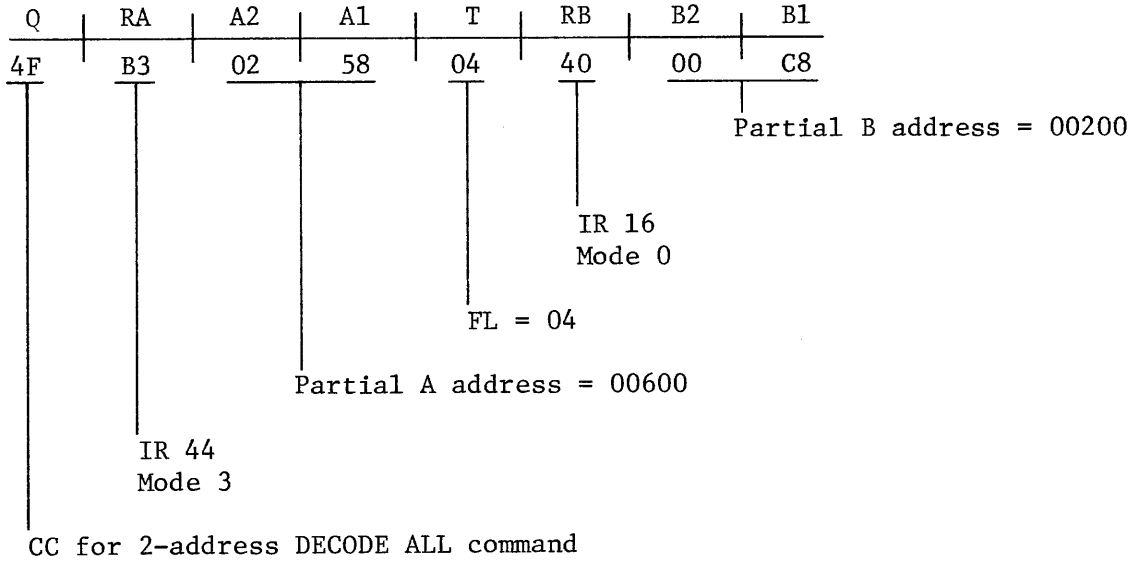
Assume the index registers contain the following:

(IR44) = 1130 (04400)
(IR16) = 0640 (01600)

After command setup:

Effective A address = 1130 + 0258 = 1388 (05000)
Effective B address = 0640 + 00C8 = 0708 (01800)

Before command execution:

A	1388	1389
(A)	0001 1100	0111 1010

BCD = 1C7A (07290)

B	0708	0709	070A	070B
(B)	0011 0011	0000 1100	0101 0011	0100 1100

BCD =   3 3   0 C   5 3   4 C

Since Mode 3 was specified, (IR44) equals 1388 (05000)

Following command execution:

B	0708	0709	070A	070B
(B)	0111 0111	0001 0010	1111 0000	1001 1001

Replacement character
from memory location
1CC6

Replacement character
from memory location
1CDD

Replacement character
from memory location
1C86

Replacement character
from memory location
1CAD
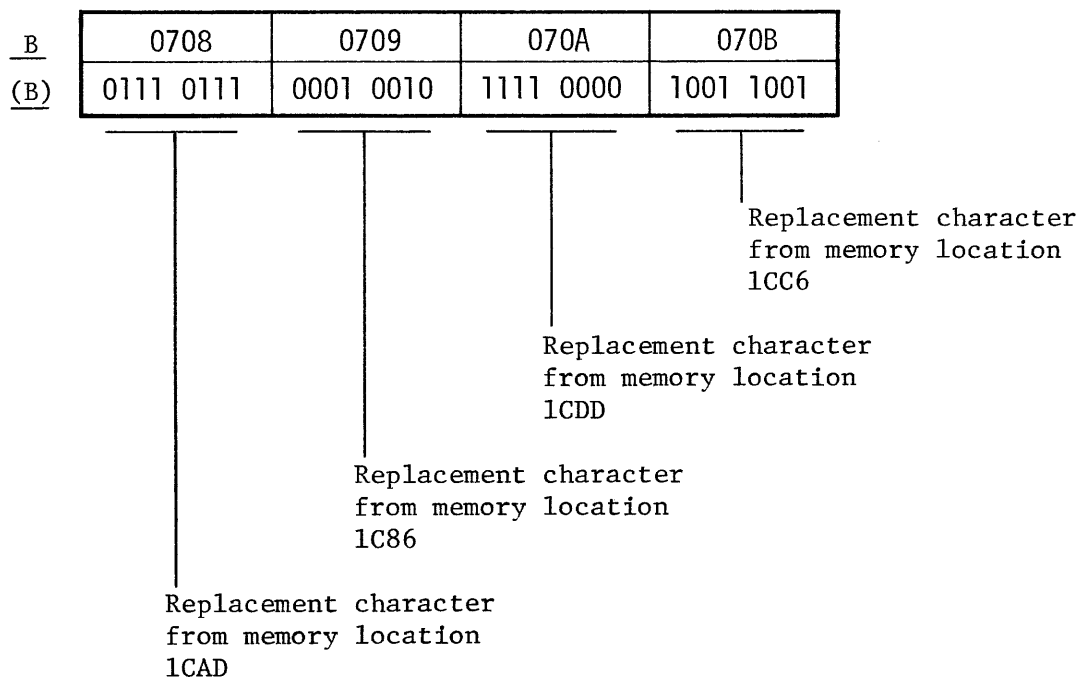
Replacement character formation (address):

1CAD
0111 0111

ADD   1C7A   contents of A(1388-89)
        33   contents of B (0708)
       1CAD   replacement characters address

1C86
0001 0010

ADD   1C7A   contents of A (1388-89)
        0C   contents of B (0709)
       1C86   replacement characters address

1CDD
1111 0000

ADD   1C7A
        53
       1CDD   replacement characters address

1CC6
1001 1001

ADD   1C7A
        4C
       1CC6   replacement characters address

The A field remains unchanged.

(IR9) equals 070C.

The G flag is ON.

C = 80(50)(D0)

The eight bits of the character designated by A are set to reflect the state of the eight option switches. Each bit is set to 1 if the corresponding option switch is ON and 0 if the corresponding option switch is OFF.

T and B are not used.

OPTION SWITCH/BIT CORRESPONDENCE								
SWITCH	8	7	6	5	4	3	2	1
BIT	B8	B7	B6	B5	B4	B3	B2	B1

Command Execution Time

E = PO + 2P

C = 81(51)(D1)

T is compared binarily to the character designated by B.

If the compared characters are identical, the repeat indicator (RI) is turned OFF and program control is transferred to A.

If A is not a legal command address, a PE occurs, and the control register is undisturbed.

If the compared characters are not identical, the next command in sequence is executed. RI is undisturbed.

The LEG flags are not used.

B = B

Command Execution Time

E = PO + P, if test is unsuccessful.

E = PO + 2P, if test is successful.

<u>C = 82(52)(D2)</u>

T is compared binarily to the character designated by B.

If the compared characters are not identical, the repeat indicator is turned OFF and program control is transferred to A. If A is not a legal command address, a PE occurs and the control register is undisturbed.

If the compared characters are identical, the next command in sequence is executed. RI is undisturbed.

The LEG flags are not used.

<u>B</u> = B

Command Execution Time

E = PO + P, if test is unsuccessful.
E = PO + 2P, if test is successful.

TEST BIT

<u>C = 83(53)(D3)</u>

B designates the character that is to be tested. The 1 bits of the T character specify which of the corresponding bits are to be tested.

If all the 1 bits of T are matched by 1 bits of the B character, or if there are no 1 bits in T, the repeat indicator is turned OFF and program control is transferred to A. If A is not a legal command address, a PE occurs and the control register is undisturbed.

If any of the 1 bits of T are matched by 0 bits in the B character, the next command in sequence is executed. RI is undisturbed.

<u>B</u> = B

Command Execution Time

E = PO + P, if test is unsuccessful.
E = PO + 2P, if test is successful.

$$C = 84(54)(D4)$$

Each character of (A) is moved into the corresponding location of (B), starting with the leftmost character.

If (A) and (B) overlap the results may be different from the results obtained when the fields do not overlap.

T specifies the length of both (A) and (B) and ranges from 0 through 255, with 0 = 256.

$\underline{B} = B + T$ unless $T = 0$, then $\underline{B} = B + 256$.

Command Execution Time

$$E = PO + 2T(P)$$

EXAMPLES

EXAMPLE 1

```
   Q  | RA | A2 | A1 |  T | RB | B2 | B1
   54 | 94 | 12 | 3C | 05 | B4 | 05 | B9
```

Partial B address = 01465

IR45
Mode 0

FL = 5

Partial A address = 04668

IR37
Mode 0

CC for 2-address MOVE A LEFT TO RIGHT command

Assume the index registers contain the following:

(IR37) = 0E74 (03700)
(IR45) = 1194 (04500)

After command setup:

Effective A address = 0E74 + 123C = 20B0 (08368)
Effective B address = 1194 + 05B9 = 174D (05965)

Before command execution:

A	20B0	20B1	20B2	20B3	20B4
(A)	0100 0011	0010 1101	0011 0110	0011 0001	0011 0101

The contents of B are not significant.

Following command execution:

(A) = (A)

B	174D	174E	174F	1750	1751
(B)	0100 0011	0010 1101	0011 0110	0011 0001	0011 0101

EXAMPLE 2 (Fields overlap)

Q	RA	A2	A1	T	RB	B2	B1
54	A0	00	00	00	A0	00	01

Partial B address = 00001

IR40
Mode 0

FL = 256

Partial A address = 00000

IR40
Mode 0

CC for 2-address MOVE A LEFT TO RIGHT command

Assume IR40 contains 1000 (04096).

After command setup:

Effective A address = 1000 + 0000 = 1000 (04096)
Effective B address = 1000 + 0001 = 1001 (04097)

Before command execution:

A	1000	1001	10EF	10FF	1100
(A)	0000 0000	0010 1100	0011 0010	1100 0100	0000 0000

Locations 1002 through 10DF

The contents of B are not significant.

When the move is initiated, the contents of 1000 replace the contents of 1001.
Because of the overlap, the character 00000000 will replace the corresponding
B-field character on each successive move.

Following command execution:

A	1000	1001	10EF	10FF	1100
(A)	0000 0000	0000 0000	0000 0000	0000 0000	0000 0000

B	1001	1002	10EF	10FF
(B)	0000 0000	0000 0000	0000 0000	0000 0000

$$C = 85(55)(D5)$$

(A) is a 1-character field called the scan key.

(B) is the field that is scanned.

T specifies the length of (B) and ranges from 0 through 255, with 0 equivalent to 256.

The scan key is compared binarily to each character of (B) starting with the leftmost character, until the scan key is less than a B character, or until (B) is exhausted.

If the scan key compares less than the B character, the address + 1 of the B character satisfying the condition is stored in IR9; the L flag (less) is turned ON, and the repeat indicator is turned OFF.

If no B character satisfies the condition, the address B + T is stored in IR9, except when T = 0. Then B + 256 is stored in IR9 and the G or E flag is set ON, depending on whether the scan key is greater than or equal to the rightmost B character. RI is undisturbed.

B = (IR9)

Command Execution Time

$$E = 2PO + 2P + 2N(P)$$

Where:

N = The number of characters scanned; $1 < N < T$

$$C = 86(56)(D6)$$

(A) is a 1-character field called the scan key.

(B) is the field that is scanned.

T specifies the length of (B) and ranges from 0 through 255, with 0 equivalent to 256.

The scan key is compared binarily to each character of (B), starting with the leftmost character, until the scan key is equal to a B character, or until (B) is exhausted.

If the scan key compares equal to the B character, the address + 1 of the B character satisfying the condition is stored in IR9. The E flag is turned ON, and the repeat indicator is turned OFF.

If no B character satisfies the condition, the address B + T is stored in IR9, except when T = 0, in which case B + 256 is stored in IR9. The G or L flag is turned ON depending on whether the scan key is greater or less than the rightmost B character. RI is undisturbed.

$\underline{B} = (\underline{IR9})$

Command Execution Time

$$E = 2PO + 2P + 2N(P)$$

Where:

N = The number of characters scanned; $1 < N < T$

$$C = 87(57)(D7)$$

(A) is a 1-character field called the scan key.

(B) is the field that is scanned.

T specifies the length of (B) and ranges from 0 through 255, with 0 equivalent to 256.

The scan key is compared binarily to each character of (B) starting with the leftmost character, until the scan key is greater than the B character, or until (B) is exhausted.

If the scan key compares greater than the B character, the address + 1 of the B character satisfying the condition is stored in IR9, the G (greater) flag is turned ON, and the repeat indicator is turned OFF.

If there is no B character that satisfies the condition, the address B + T is stored in IR9, except when T = 0, then B + 256 is stored in IR9. The E or L flag is set ON depending on whether or not the scan key is equal to or less than the rightmost B character. RI is undisturbed.

$\underline{B}$ = ($\underline{IR9}$)

Command Execution Time

$$E = 2PO + 2P + 2N(P)$$

Where:

   N = The number of characters scanned; $1 < N < T$

After command setup assume that:

Effective A address = 0800 (02048)
Effective B address = 04B0 (01200)

T = 4

SCAN ON KEY LESS THAN

Assume that the A and B fields contain the following information:

A	0800
(A)	0011 0100

ASCII = 4 (Scan Key)

B	04B0	04B1	04B2	04B3
(B)	0011 0100	0011 0011	0011 0110	0011 0001

ASCII = 4   3   6   1

Following command execution:

Since the contents of 04B2 are greater than the scan key, the scan is satisfied
and the address 04B3 is stored in IR9.

SCAN ON KEY EQUAL

The A and B fields of the above example are used.

Following command execution:

Since the contents of 04B0 are equal to the scan key, the scan is satisfied
and the address 04B1 is stored in IR9.

SCAN ON KEY GREATER THAN

The A and B fields of the above examples are used.

Following command execution:

Since the contents of 04B1 are less than the scan key, the scan is satisfied
and the address 04B2 is stored in IR9.

<u>NOTE</u>

In any of the above examples, if the scan key (A field)
were such that no B field character satisfied the scan,
then (IR9) = 04B4. The A and B fields are unchanged in
all cases.

C = 96(60)(E0)

The contents of the field specified by the effective A address are added binarily to the contents of the field specified by the effective B address. The result replaces the contents of the B field.

The addition is performed from right to left, 8 bits (one byte) at a time. A carry from one byte is added to the next byte to the left; a carry from the leftmost byte is ignored (the overflow flag is not affected). Both fields are considered unsigned and positive.

The T portion of the command designates the number of bytes in each field. T may be any value from 0 through 255, with 0 equivalent to 256.

Following command execution, the A field retains its original contents. If the A and B fields overlap, the results may be different from the results obtained when the fields do not overlap. In this case, the initial contents of the A field are altered.

B = B

Command Execution Time

$E = 2PO + 2P + 3T(P)$

EXAMPLE 1

Q	RA	A2	A1	T	RB	B2	B1
60	03	08	00	03	30	00	00

Partial B address – 00000

IR12
Mode 0

FL = 3

Partial A address – 02048

IR(none)
Mode 3

CC for 2-address ADD BINARY command

Assume IR12 contains 04B0 (01200)

After command setup:

Effective A address = 0800 (02048).   Since no indexing is required, the specification of mode 3 is meaningless in this case.

Effective B address = 04B0 + 0000 = 04B0 (01200)

Before command execution:

A	0800	0801	0802
(A)	1010 1100	1100 1010	0010 1101

B	04B0	04B1	04B2
(B)	1110 0110	1011 1110	0111 0101

```
(A)  10101100    11001010    00101101
(B)  11100110    10111110    01110101

     01001010    01110100    01011000    Partial sum
   1  1  1  1       1 1       1  1 1      Carries

   1⌐10010011    10001000    10100010    Final sum
```
The carry beyond the specified field length is ignored.

Following command execution:

(A) = Unchanged

B	04B0	04B1	04B2
(B)	1001 0011	1000 1000	1010 0010

B = 04B0 (01200)

EXAMPLE 2 (Fields Overlap)

```
Q  |  RA  |  A2  |  A1
EO |  20  |  09  |  00
|     |       |
|     |       Partial A address = 02304
|     |
|     IR8
|     Mode 0
|
CC for 1-address ADD BINARY command
```
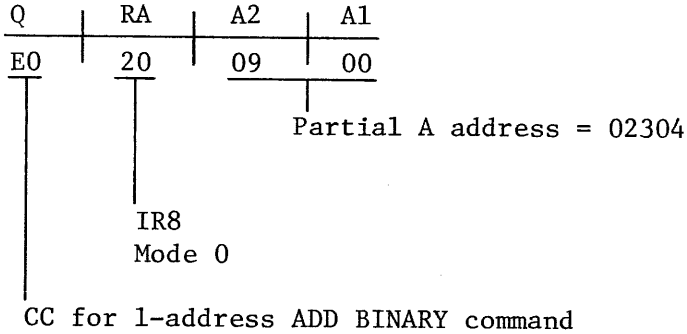
Assume IR8 contains FFFD (65,533). Since indexing on the NCR Century 200 is cyclic:

Effective A address = FFFD + 0900 = 08FD (02301).

Since a 1-address command is specified, the B and T values stored from a previous command are used. Assume that the T and B registers contain 04 and 08FC (02300), respectively.

Before command execution:

A	08FD	08FE	08FF	0900
(A)	0001 0100	0010 1011	0011 1101	1111 0010

B	08FC	08FD	08FE	08FF
(B)	0000 0000	0001 0100	0010 1011	0011 1101

Because the fields overlap, addition takes place as follows:

1. The contents of 0900 are added to the contents of 08FF, and the result is stored in 08FF.

```
11110010
00111101
11001111   Partial sum
11         Carries
1 00101111 Final sum
 Carry to next byte
```

2. The contents of 08FF (altered by step 1) are added to the contents of 08FE, and the result is stored in 08FE.

```
00101111
00101011
00000100   Partial sum
 1 1 111   Carries (in-
01011011   cluding initial
           carry from
           step 1).
```

3. The contents of 08FE (altered by step 2) are added to the contents of 08FD, and the result is stored in 08FD.

```
01011011
00010100
01001111   Partial sum
1          Carry
01101111   Final sum
```

4. The contents of 08FD (altered by step 3) are added to the contents of 08FC, and the result is stored in 08FC.

```
01101111
00000000
01101111   Partial & final
           sum
```

Following command execution:

A	08FD	08FE	08FF	0900
(A)	0110 1111	0101 1011	0010 1111	1111 0010

B	08FC	08FD	08FE	08FF
(B)	0110 1111	0110 1111	0101 1011	0010 1111

B = 08FC (02300)

$$C = 97(61)(E1)$$

The contents of the field specified by the effective A address are subtracted binarily from the contents of the field specified by the effective B address; the results replace the initial contents of the B field.

Subtraction is performed from right to left, one character (byte) at a time. As explained in the NCR Century 200 PROCESSOR manual, subtraction is actually performed by complementary addition. The 1's complement of the A character is formed and added to the B character to derive a partial sum. Carries from the first addition and an initial carry (to compensate for use of the 1's complement) are added to the partial sum to derive the final result. A carry beyond the leftmost character position in the specified field is ignored (the overflow flag is not affected). Both A and B fields are considered unsigned and positive.

The T portion of the command specifies the number of characters in each field; T may specify any number from 0 through 255, with 0 equivalent to 256.

At completion of the command, the A field retains its initial contents. The initial contents of the B field are replaced by the results of the subtraction. If the contents of the A field are greater than the contents of the B field, the final result stored is the 2's complement of (A) − (B).

If the A and B fields overlap, the results may be different from the results obtained when the fields do not overlap. The initial contents of the A field will also change.

$\underline{B} = B$

Command Execution Time

$E = 2PO + 2P + 3T(P)$

EXAMPLE 1

Q	RA	A2	A1	T	RB	B2	B1
61	00	01	2C	02	00	09	00

Partial B address = 02304

IR (none)
Mode 0

FL = 2

Partial A address = 00300

IR (none)
Mode 0

CC for 2-address SUBTRACT BINARY Command
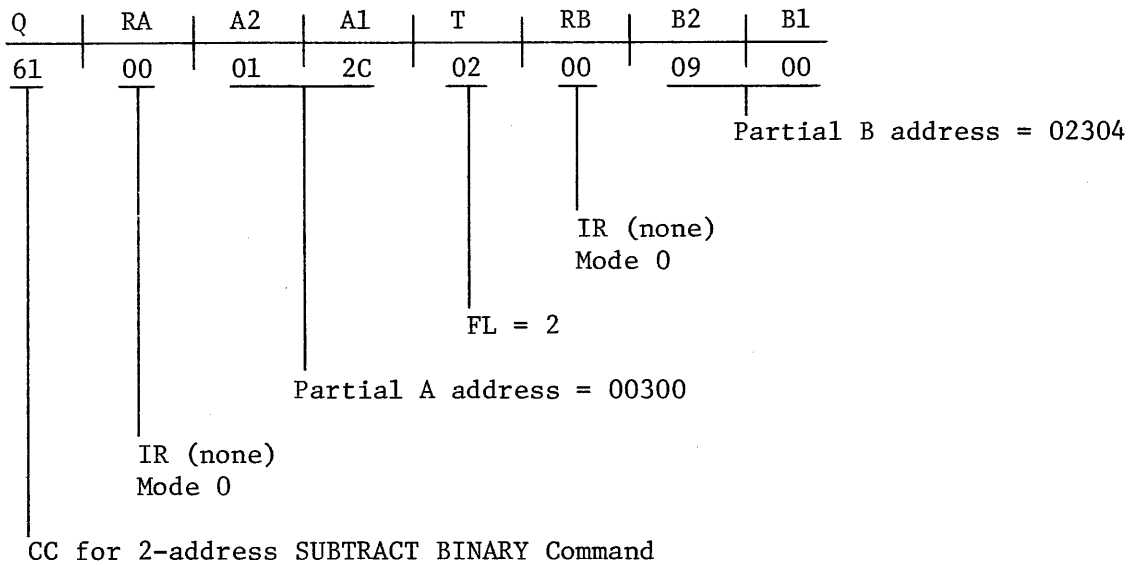
After command setup:

Effective A address = 012C (00300)
Effective B address = 0900 (02304)

Before command execution:

A	012C	012D
(A)	0000 0000	0010 1000

B	0900	0901
(B)	0001 0010	0110 1100

Subtraction:

```
    11111111    11010111    1's complement of (A)
    00010010    01101100
    11101101    10111011    Partial Sum
      1  1        1  1      Carries
                       1    Initial Carry to form 2's complement
  1 01010010    01000100
```

Carry beyond the specified field length is ignored.

After command execution:

(A) - unchanged

B	0900	0901
(B)	0101 0010	0100 0100

EXAMPLE 2 (Contents of effective A address > contents of effective B address)

Q	RA	A2	A1	T	RB	B2	B1
61	58	03	20	02	78	00	C8

Partial B address = 00200

IR30
Mode 0

FL = 2

Partial A address = 00800
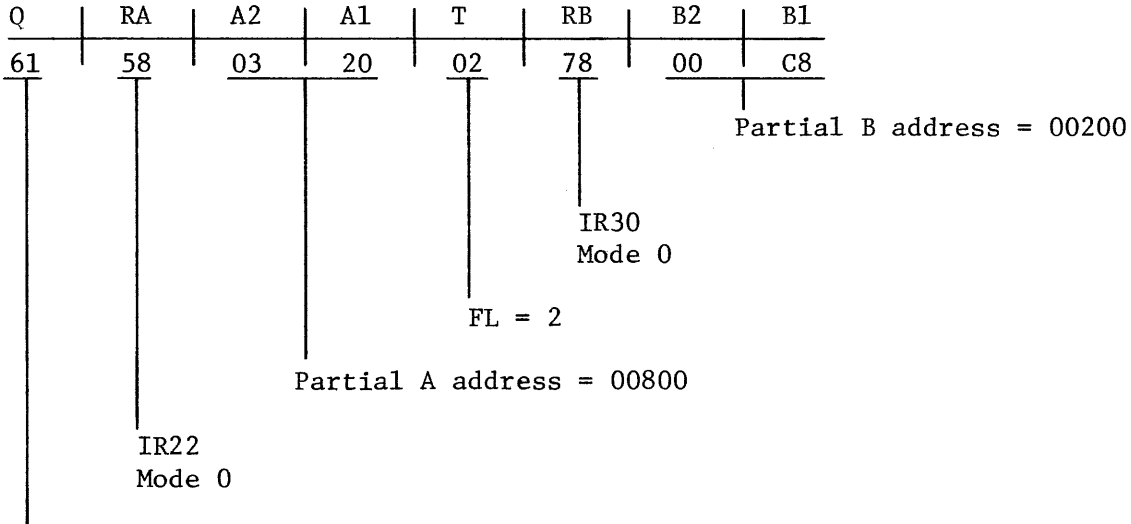
IR22
Mode 0

CC for 2-address SUBTRACT BINARY command

Assume that the index registers contain the following:

(IR22) = 0898 (02200)
(IR30) = 0BB8 (03000)

After command setup:

Effective A address = 0898 + 0320 = 0BB8 (03000)
Effective B address = 0BB8 + 00C8 = 0C80 (03200)

Before command execution:

A	0BB8	0BB9
(A)	0111 0101	1010 0100

B	0C80	0C81
(B)	0100 0011	0110 1101

Subtraction:

10001010	01011011	1's complement of A
01000011	01101101	
11001001	00110110	Partial Sum
1	1 1 1	Carries
	1	Initial Carry
11001101	11001001	

Following command execution:

(A) – unchanged

B	0C80	0C81
(B)	1100 1101	1100 1001

C = 98(62)(E2)

The contents of the field specified by the effective A address are added decimally to the contents of the field specified by the effective B address (see addition tables, page 61). The result of the addition replaces the contents of the B field.

Decimal (BCD) addition is performed from right to left, one byte at a time. Only b4 - b1 of each byte are added. The four most-significant bits of the byte are ignored, but the configuration 0011 is stored in b8 - b5 of each byte in the result.

Effectively, each A field character is added to excess six (see BCD Arithmetic section of NCR Century 200 PROCESSOR manual) and then added to the corresponding B field character. If the addition of the B field character causes a carry beyond the four bits, the result is stored in b4 - b1 of the B field character and the carry is added to the next character position to the left. If no carry occurs, logic circuitry corrects for the excess six and stores the four corrected bits in b4 - b1 of the corresponding B field character. A carry beyond the leftmost position of the defined field is discarded, but the overflow flag (OF) is turned ON.

The T portion of the command designates the number of characters in each operand field; T may be any value from 0 through 255, with 0 equivalent to 256.
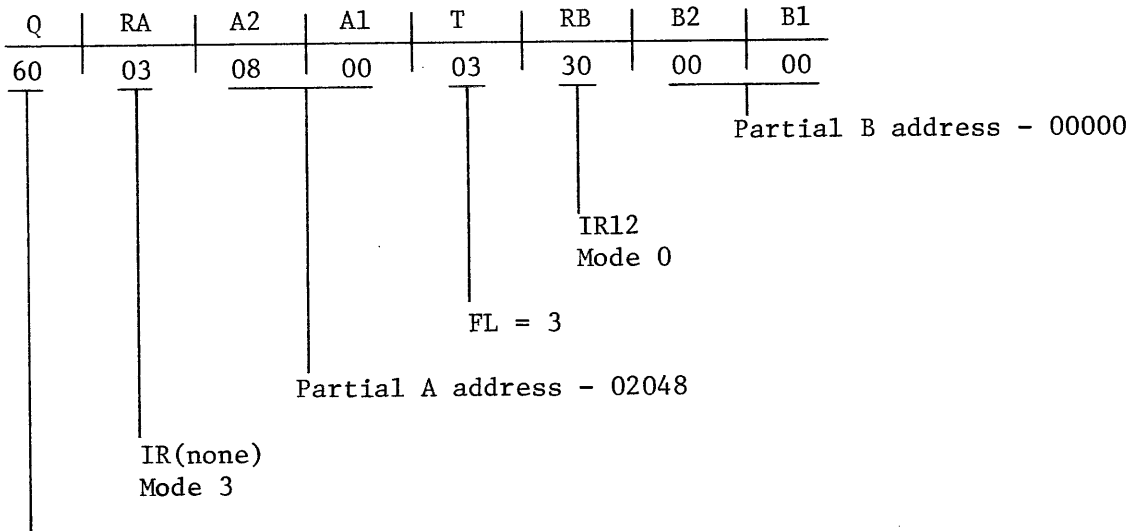
On completion of the ADD UNSIGNED command, the A field retains its initial contents; the initial contents of the B field are replaced by (A) + (B).

If the A and B fields overlap, the results may be different from the results obtained where the fields do not overlap. In this case, the initial contents of the A field are changed.

Packed data must be unpacked before execution of this command.

B = B

Command Execution Time

$E = 2PO + 2P + 3T(P)$

ADDITION TABLES

NO CARRY FROM PREVIOUS DIGIT POSITION
DIGITS FROM (A)

(B) \ (A)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	c0	B	C	D	E	F	0
2	2	3	4	5	6	7	8	9	c0	c1	C	D	E	F	0	1
3	3	4	5	6	7	8	9	c0	c1	c2	D	E	F	0	1	2
4	4	5	6	7	8	9	c0	c1	c2	c3	E	F	0	1	2	3
5	5	6	7	8	9	c0	c1	c2	c3	c4	F	0	1	2	3	4
6	6	7	8	9	c0	c1	c2	c3	c4	c5	0	1	2	3	4	5
7	7	8	9	c0	c1	c2	c3	c4	c5	c6	1	2	3	4	5	6
8	8	9	c0	c1	c2	c3	c4	c5	c6	c7	2	3	4	5	6	7
9	9	c0	c1	c2	c3	c4	c5	c6	c7	c8	3	4	5	6	7	8
A	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	4	5	6	7	8	9
B	c1	c2	c3	c4	c5	c6	c7	c8	c9	cA	5	6	7	8	9	c0
C	c2	c3	c4	c5	c6	c7	c8	c9	cA	cB	6	7	8	9	c0	c1
D	c3	c4	c5	c6	c7	c8	c9	cA	cB	cC	7	8	9	c0	c1	c2
E	c4	c5	c6	c7	c8	c9	cA	cB	cC	cD	8	9	c0	c1	c2	c3
F	c5	c6	c7	c8	c9	cA	cB	cC	cD	cE	9	c0	c1	c2	c3	c4

DIGITS FROM (B)

CARRY FROM PREVIOUS DIGIT-POSITION
DIGITS FROM (A)

(B) \ (A)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	1	2	3	4	5	6	7	8	9	c0	B	C	D	E	F	0
1	2	3	4	5	6	7	8	9	c0	c1	C	D	E	F	0	1
2	3	4	5	6	7	8	9	c0	c1	c2	D	E	F	0	1	2
3	4	5	6	7	8	9	c0	c1	c2	c3	E	F	0	1	2	3
4	5	6	7	8	9	c0	c1	c2	c3	c4	F	0	1	2	3	4
5	6	7	8	9	c0	c1	c2	c3	c4	c5	0	1	2	3	4	5
6	7	8	9	c0	c1	c2	c3	c4	c5	c6	1	2	3	4	5	6
7	8	9	c0	c1	c2	c3	c4	c5	c6	c7	2	3	4	5	6	7
8	9	c0	c1	c2	c3	c4	c5	c6	c7	c8	3	4	5	6	7	8
9	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	4	5	6	7	8	9
A	c1	c2	c3	c4	c5	c6	c7	c8	c9	cA	5	6	7	8	9	c0
B	c2	c3	c4	c5	c6	c7	c8	c9	cA	cB	6	7	8	9	c0	c1
C	c3	c4	c5	c6	c7	c8	c9	cA	cB	cC	7	8	9	c0	c1	c2
D	c4	c5	c6	c7	c8	c9	cA	cB	cC	cD	8	9	c0	c1	c2	c3
E	c5	c6	c7	c8	c9	cA	cB	cC	cD	cE	9	c0	c1	c2	c3	c4
F	c6	c7	c8	c9	cA	cB	cC	cD	cE	cF	c0	c1	c2	c3	c4	c5

DIGITS FROM (B)

EXAMPLE 1

Q	RA	A2	A1	T	RB	B2	B1
62	00	27	10	04	EC	02	58

Partial B address = 00600

IR59
Mode 0

FL = 4

Partial A address = 10000

IR (none)
Mode 0

CC for 2-address ADD UNSIGNED command
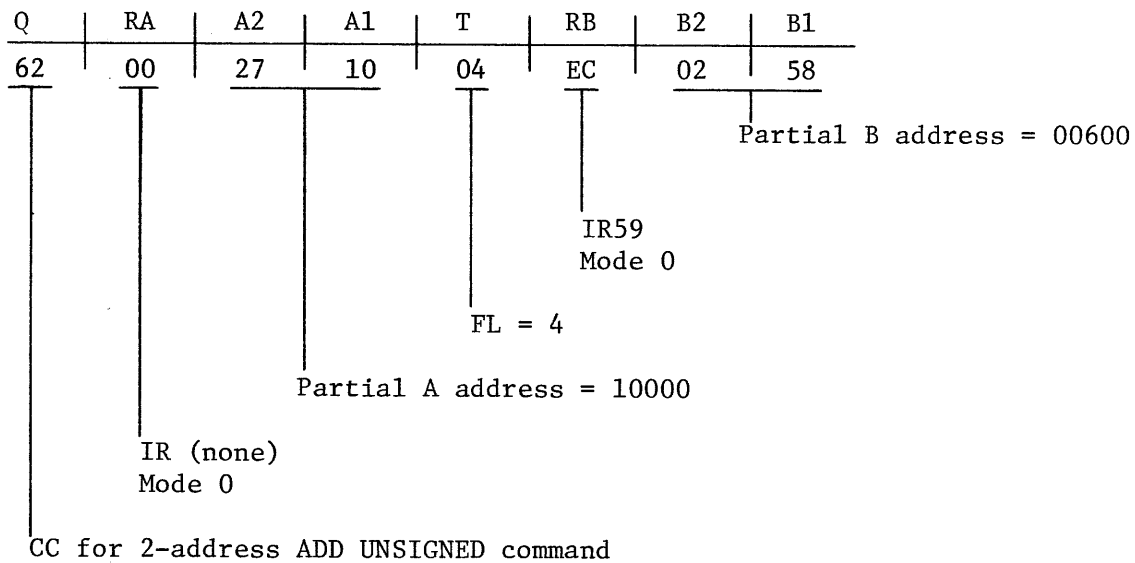
Assume IR59 contains 170C (05900)

After command setup:

Effective A address = 2710 (10000) - no indexing required
Effective B address = 170C + 0258 = 1964 (06500)

Before command execution:

A	2710	2711	2712	2713
(A)	0011 0100	0011 0111	0011 0010	0011 1000

B	1964	1965	1966	1967
(B)	0011 0110	0011 0100	0011 0100	0011 0011

Addition:

Decimal equivalent of addition to be performed:

$$
\begin{array}{r}
4728 \\
+\ \ 6443 \\
\hline
11171
\end{array}
$$

1. Excess six is added to b4 - b1 of each A field character.

0100	0111	0010	1000	
0110	0110	0110	0110	
0010	0001	0100	1110	Partial sum
1	11	1		Carries
1010	1101	1000	1110	Final sum

2. B4 - b1 of each corresponding B field character are added to the result obtained in step 1:

	1010	1101	1000	1110	
	0110	0100	0100	0011	
	1100	1001	1100	1101	Partial sum
	1	1		1	Carries
1	0001	0001	1101	0001	Uncorrected final sum

↖Carry sets OF ON

3. Since 1101 is not a valid BCD number (its binary value 9), hardware makes a decimal correction by, in effect, adding the 2's complement of 6:

$$
\begin{array}{r}
1101 \\
1010 \\
\hline
1\ \ 0111
\end{array}
$$

↖Carry is ignored

4. The corrected final sum is then stored in b4 - b1 of the corresponding B-field characters, with zone bits 0011 stored in b8 - b5 of each character:

	0001	0001	0111	0001
B	1964	1965	1966	1967
(B)	0011 0001	0011 0001	0011 0111	0011 0001
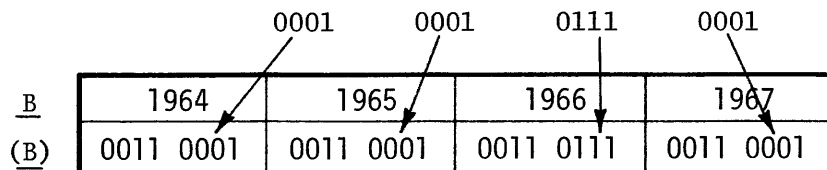
Decimal value in NCR Century ASCII code = 1171, with OF ON, indicating overflow carry of 1.

EXAMPLE 2 (Fields Overlap)
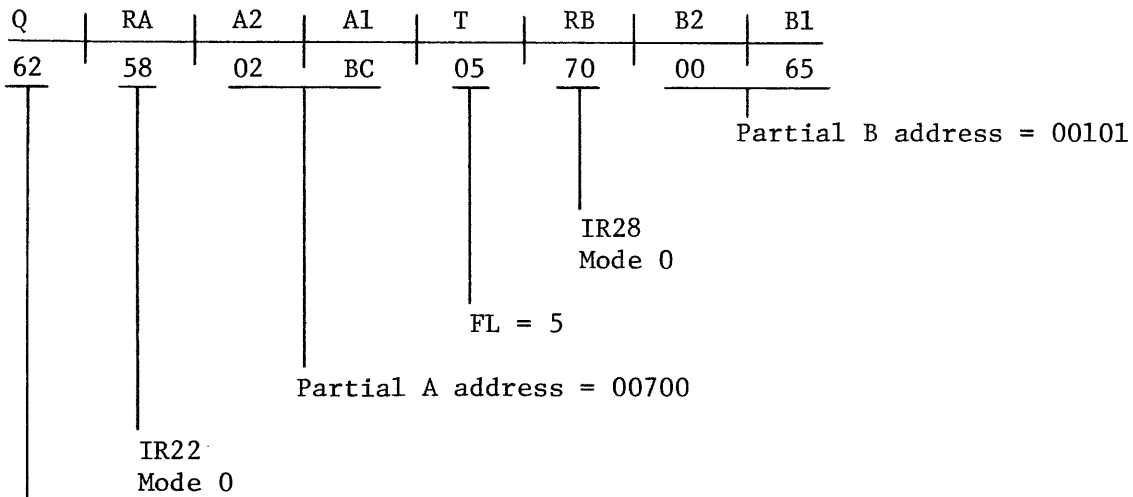
Q	RA	A2	A1	T	RB	B2	B1
62	58	02	BC	05	70	00	65

Partial B address = 00101

IR28
Mode 0

FL = 5

Partial A address = 00700

IR22
Mode 0

CC for 2-address ADD UNSIGNED command.

Assume the index registers contain the following:

(IR22) = 0898 (02200)
(IR28) = 0AF0 (02800)

After command setup:

Effective A address = 0898 + 02BC = 0B54 (02900)
Effective B address = 0AF0 + 0065 = 0B55 (02901)

Before command execution:

0B54	0B55	0B56	0B57	0B58	0B59
0011 0100	0011 0001	0011 0010	0011 0110	0011 1001	0011 0000
	Overlap Area				

Addition:

Decimal equivalent of addition to be performed:

$$41269$$
$$+\ \underline{12690}$$
$$53959$$

1. Excess six is added to b4 - b1 of each A-field character:

0100	0001	0010	0110	1001	
0110	0110	0110	0110	0110	
0010	0111	0100	0000	1111	Partial sum
1		1	11		Carries
1010	0111	1000	1100	1111	Final sum

2. B4 - b1 of each corresponding B-field character are added to the results obtained in step 1:

1010	0111	1000	1100	1111	
0001	0010	0110	1001	0000	
1011	0101	1110	0101	1111	Partial sum
	1	1			Carries
1011	1001	1111	0101	1111	Final sum

3. The sums of those additions which resulted in an illegal BCD number
   (binary value > 9) or which did not generate a carry beyond b4 are
   decimally corrected by, in effect, subtracting the excess six:

1011	1001	1111	0101	1111
0110	0110	0110		0110
0101	0011	1001	0101	1001

4. The corrected results replace b4 - b1 of each corresponding B character.
   Note that, in this case, the overlap did not cause the sum to be any
   different from the sum that would be obtained if the fields did not
   overlap. Only (A) is affected.

Following command execution:

A	0B54	0B55	0B56	0B57	0B58
(A)	0011 0100	0011 0101	0011 0011	0011 1001	0011 0101

B	0B55	0B56	0B57	0B58	0B59
(B)	0011 0101	0011 0011	0011 1001	0011 0101	0011 1001

BCD
value =
53959

$$\underline{C = 99(63)(E3)}$$

The contents of the field specified by the effective A address are subtracted decimally from the contents of the field specified by the effective B address (see subtraction tables, page 67). The result of the subtraction replaces the contents of the B field.

Decimal (BCD) subtraction is performed from right to left, one byte at a time. Only b4 - b1 of each byte are used. The four most-significant bits of each byte are ignored, but the zone-bit configuration 0011 is stored in b8 - b5 of each byte in the result.

The 1's complement of the A-bit configuration is formed and added to the B-bit configuration to derive a partial sum. Carries from the first addition and an initial carry (to compensate for the use of the 1's complement) are added to the partial sum to derive the final sum. A carry beyond the leftmost bit position is not included in the result, but is used by the processor to determine whether the result was positive (contents of B > contents of A) or negative (contents of A > contents of B). If there is a carry beyond the leftmost bit position, the result is positive and is stored in b4 - b1 of the corresponding B character. No carry indicates that the result is negative and the processor makes a decimal correction (in effect, subtracting excess six from the result) and stores the 10's complement of (A) - (B). If (A) > (B), the overflow indicator is turned ON.
The T portion of the command specifies the number of characters in each field; T may be any number from 0 through 255, with 0 equivalent to 256.

At completion of command execution, the A field retains its initial contents. The initial contents of the B field are replaced by the result of the subtraction.

If the A and B fields overlap, the results may be different from the results of an operation where the fields do not overlap. The initial contents of the A field will also be altered.

Packed fields must be unpacked before executing this command.

$$\underline{B = B}$$

Command Execution Time

$$E = 2PO + 2P + 3T \ (P)$$

SUBTRACTION TABLES

NO BORROW BY PREVIOUS DIGIT-POSITION
DIGITS FROM (A)

DIGITS FROM (B)

(B)\(A)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2	ʙ1	ʙ0	ʙF	ʙE	ʙD	ʙC	ʙB
1	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2	ʙ1	ʙ0	ʙF	ʙE	ʙD	ʙC
2	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2	ʙ1	ʙ0	ʙF	ʙE	ʙD
3	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2	ʙ1	ʙ0	ʙF	ʙE
4	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2	ʙ1	ʙ0	ʙF
5	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2	ʙ1	ʙ0
6	6	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2	ʙ1
7	7	6	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2
8	8	7	6	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3
9	9	8	7	6	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4
A	A	9	8	7	6	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5
B	B	A	9	8	7	6	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6
C	C	B	A	9	8	7	6	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7
D	D	C	B	A	9	8	7	6	5	4	3	2	1	0	ʙ9	ʙ8
E	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	ʙ9
F	F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0

BORROW BY PREVIOUS DIGIT-POSITION
DIGITS FROM (A)

DIGITS FROM (B)

(B)\(A)	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2	ʙ1	ʙ0	ʙF	ʙE	ʙD	ʙC	ʙB	ʙA
1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2	ʙ1	ʙ0	ʙF	ʙE	ʙD	ʙC	ʙB
2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2	ʙ1	ʙ0	ʙF	ʙE	ʙD	ʙC
3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2	ʙ1	ʙ0	ʙF	ʙE	ʙD
4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2	ʙ1	ʙ0	ʙF	ʙE
5	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2	ʙ1	ʙ0	ʙF
6	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2	ʙ1	ʙ0
7	6	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2	ʙ1
8	7	6	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3	ʙ2
9	8	7	6	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4	ʙ3
A	9	8	7	6	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5	ʙ4
B	A	9	8	7	6	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6	ʙ5
C	B	A	9	8	7	6	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7	ʙ6
D	C	B	A	9	8	7	6	5	4	3	2	1	0	ʙ9	ʙ8	ʙ7
E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	ʙ9	ʙ8
F	E	D	C	B	A	9	8	7	6	5	4	3	2	1	0	ʙ9

EXAMPLE 1

Q	RA	A2	A1	T	RB	B2	B1
63	90	00	64	04	B0	00	00

Partial B address = 00000

IR44
Mode 0

FL = 4

Partial A address = 00100

IR36
Mode 0

CC for 2-address SUBTRACT UNSIGNED Command.

Assume the index registers contain the following:

 (IR36) = 0E10 (03600)
 (IR44) = 1130 (04400)

After command setup:

Effective A address = 0E10 + 0064 = 0E74 (03700)
Effective B address = 1130 + 0000 = 1130 (04400)

Before command execution:

A	0E74	0E75	0E76	0E77
(A)	0011 0100	0011 0100	0011 0010	0011 1000

B	1130	1131	1132	1133
(B)	0011 0111	0011 0010	0011 0011	0011 1001

Subtraction:

Decimal equivalent of the subtraction to be performed:

```
     7239
  -  4428
     2811
```

1. As each A-field configuration is read in, it is converted to its 1's complement:

A-field Bits	1's Complement
0100	1011
0110	1011
0010	1101
1000	0111

2. The complements are then added to the corresponding B characters:

```
   1011    1011    1101    0111
   0111    0010    0011    1001
   1100    1001    1110    1110    Partial sum
   11      1       1       1       Carries
      1       1       1       1    Initial carries
c  0010   c1110   c0001   c0001    Uncorrected result
```

This carry not stored

3. Since there was no carry from third set of bits added, their sum must be corrected:

```
   0010    1110    0001    0001    Uncorrected result
           0110                    Excess six correction
   0010    1000    0001    0001    Corrected result
```

4. The results are then stored in b4 - b1 of the corresponding B-field characters, and b8 - b5 of each character are set to 0011:

B	1130	1131	1132	1133
(B)	0011 0010	0011 1000	0011 0001	0011 0001

BCD value = 2811

EXAMPLE 2 (A) > (B)

Q	RA	A2	A1	T	RB	B2	B1
63	74	01	4E	03	74	10	C2

Partial B address = 04290

IR29
Mode 0

FL = 3

Partial A address = 00334

IR29
Mode 0

CC for 2-address SUBTRACT UNSIGNED command
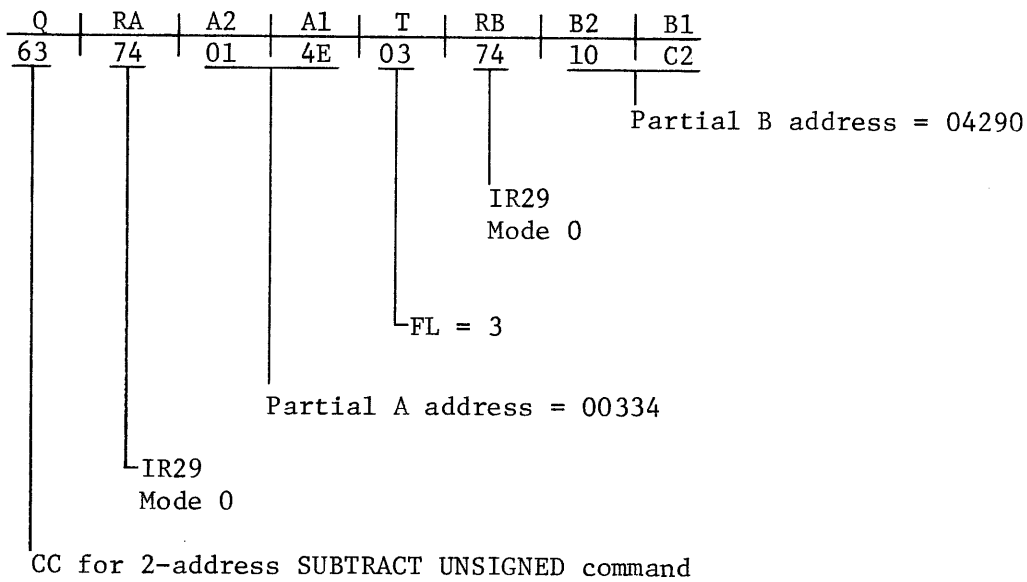
Assume IR29 contains 067E (01662).

After command setup:

Effective A address = 067E + 014E = 07CC (01996)
Effective B address = 067E + 10C2 = 1740 (05952)

Before command execution:

A	07CC	07CD	07CE
(A)	0011 0101	0011 0100	0011 0010

B	1740	1741	1742
(B)	0011 0100	0011 0110	0011 0100

Subtraction :

   Decimal equivalent of subtraction to be performed:

                464
              - 542
              - 078

Since the result was negative, the 10's complement of (A) - (B) is stored:

                542   1000
              - 464   - 078
                078   922   10's complement of 078

1. The 1's complement is formed for each set of A-field bits:

        A-field Bits   1's Complement

           0101   1010
           0100   1011
           0010   1101

2. The complement A-field bits are added to the corresponding B-field bits:

        1010   1011   1101
        0100   0110   0100
        1110   1101   1001   Partial sum
                  1   1   Carries
           1   1   1   Initial carries
        1111   c0010   c0010   Uncorrected result

3. Since there was no carry beyond b4 of the leftmost digit position, the
   four leftmost bits must be corrected:

        1111   0010   0010   Uncorrected result
        0110   Excess six
        1001   0010   0010   Corrected result

4. The corrected results are stored in b4 - b1 of the corresponding B-field
   characters, and b8 - b5 of each character are set to 0011:

   B | 1740 | 1741 | 1742 | BCD value = 922 = 10's
   (B) | 0011 1001 | 0011 0010 | 0011 0010 | complement of (A) - (B)
                                                     OI is turned ON.

$$C = 100(64)(E4)$$

The contents of the field specified by the effective A address are moved into the field specified by the effective B address. Moving is done from right to left, one character at a time, with each A-field character replacing the corresponding B-field character.

The T portion of the command specifies the number of characters to be moved, and may range in value from 0 through 255, with 0 equivalent to 256.

At the completion of the command, the A field retains its initial contents. If the A and B fields overlap, the results may be different from the operation where the fields do not overlap. In this case, the initial contents of the A field are always changed.

$\underline{B} = B$

Command Execution Time

$E = 2PO + 2P + 2T(P)$

EXAMPLES

EXAMPLE 1

```
 Q | RA | A2 | A1 | T | RB | B2 | B1
64 | 94 | 12 | 3C | 05| B4 | 05 | B9
```

Partial B address = 01465

IR45
Mode 0

FL = 5

Partial A address = 04668

IR37
Mode 0

CC for 2-address MOVE A RIGHT TO LEFT command

Assume the index registers contain the following:

(IR37) = OE74 (03700)
(IR45) = 1194 (04500)

After command setup:

Effective A address = OE74 + 123C = 20B0   (08368)
Effective B address = 1194 + 05B9 = 174D   (05965)

Before command execution:

A	20B0	20B1	20B2	20B3	20B4
(A)	0100 0011	0010 1101	0011 0110	0011 0001	0011 0101

The contents of B are not significant.

Following command execution:

(A) = (A)

B	174D	174E	174F	1750	1751
(B)	0100 0011	0010 1101	0011 0110	0011 0001	0011 0101

EXAMPLE 2   (Fields Overlap)

Q	RA	A2	A1	T	RB	B2	B1
64	A0	00	01	00	A0	00	00

Partial B address = 00000

IR40
Mode 0

FL = 256

Partial A address = 00001

IR40
Mode 0

CC for 2-address MVAR command

Assume IR40 contains 1000 (04096).

After command setup:

Effective A address = 1000 + 0001 = 1001 (04097)
Effective B address = 1000 + 0000 = 1000 (04096)

Before command execution:

A	1001	1002	//	10EF	10FF	1100
(A)	1111 1011	0010 1100	//	0011 0010	1100 0100	0000 0000

Locations 1003 through 10DF

The contents of B are not significant.

When the move is initiated, the contents of 1100 replace the contents of 10FF. Because of the overlap the character 00000000 will replace the corresponding B-field character on each successive move.

Following command execution:

A	1001	1002	//	10EF	10FF	1100
(A)	0000 0000	0000 0000	//	0000 0000	0000 0000	0000 0000

B	1000	1001	//	10EF	10FF
(B)	0000 0000	0000 0000	//	0000 0000	0000 0000

C = 101(65)(E5)

The contents of the field specified by the effective A address are compared binarily to the contents of the field specified by the effective B address, and a G, L, or E flag is turned ON to indicate the result of the comparison.

The binary comparison is performed one character at a time, from right to left. Effectively, the complement of the A-field character is added to the B-field character, as in a binary subtraction operation. If the contents of A are less than the contents of B, the addition results in a carry beyond the leftmost character in the field, and the number unequal indicator is turned ON. This combination causes the L flag to be set ON.

If the contents of A and B are equal, the addition also results in a carry beyond the leftmost character in the field, but the number unequal indicator is turned OFF or remains OFF. This combination sets the E flag ON.

If the contents of A are greater than the contents of B, no carry occurs beyond the leftmost character position. This causes the G flag to be set ON.

The conditions (A) > (B) and (A) = (B) also cause the repeat indicator (RI) to be turned OFF. The condition (A) < (B) does not affect the status of the RI.

T may range in value from 0 through 255, with 0 equivalent to 256.

$\underline{B} = B$

Command Execution Time

$$E = 2 \text{ PO} + 2P + 3T\,(P)$$

EXAMPLE 1   (A) < (B)

Q	RA	A2	A1	T	RB	B2	B1
65	00	02	00	03	00	03	00

Partial B address = 00768

IR (none)
Mode 0

FL = 3

Partial A address = 00512

IR (none)
Mode 0

CC for 2-address COMPARE BINARY command
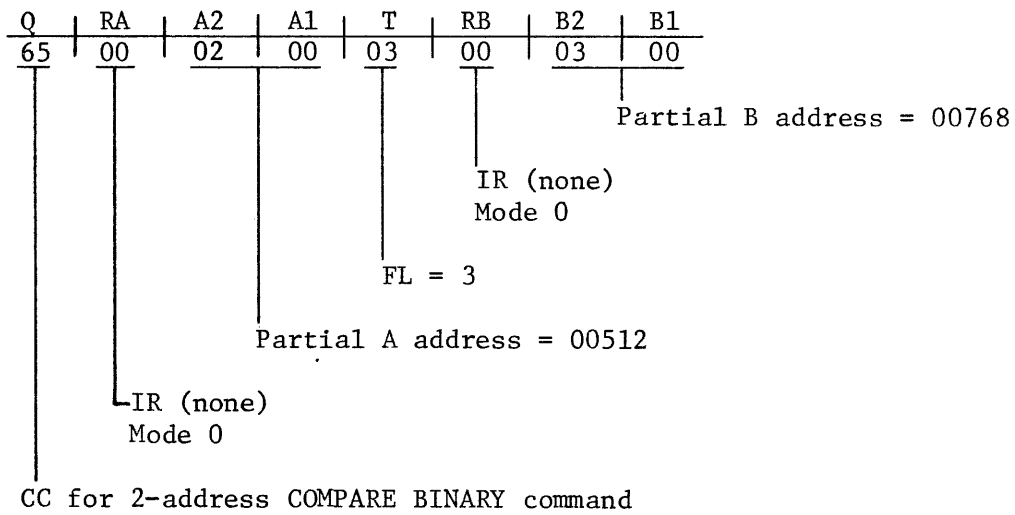
After command setup:

Effective A address = 0200 (00512) - No indexing required.
Effective B address = 0300 (00768) - No indexing required.

Before command execution:

A	0200	0201	0202
(A)	0011 0111	0011 0110	0011 1001

B	0300	0301	0302
(B)	0011 1001	0011 0100	0011 1001

Binary Comparison:

1. The 1's complement is derived for each A-field character:

A-field Character   1's Complement of A-field Character

   00110111   11001000
   00110110   11001001
   00111001   11000110

2. The 1's complements of the A-field characters are added to the corresponding B-field characters:

```
        11001000    11001001    11000110
        00111001    00110100    00111001

        11110001    11111101    11111111    Partial sum
            1                               Carry
                                     1      Initial carry
     1  00000001    11111110    00000000    Final sum
```

3. Because the result is not equal to zero, the number unequal indicator is turned ON.

4. Because there is a carry beyond the leftmost character in the specified field, and the number unequal indicator is ON, the L flag is set ON.

EXAMPLE 2 (A) = (B)

Assume that the fields defined in example 1 had contained the following:

A	0200	0201	0202
(A)	0100 0001	0011 1101	0100 0010

B	0300	0301	0302
(B)	0100 0001	0011 1101	0100 0010

Binary comparison:

```
1's Complement of (A)   10111110    11000010    10111101
                (B)     01000001    00111101    01000010

                        11111111    11111111    11111111    Partail sum
                                                     1      Initial carry
                     1  00000000    00000000    00000000    Final sum
```

Conditions:

1. Number unequal indicator is turned OFF or remains OFF since the final sum in the three specified fields is zero.

2. The carry beyond the specified field length and the condition of the number unequal indicator cause the E flag to be set ON and the repeat indicator to be turned OFF if it is ON.

Example 3.   (A) > (B)

Assume that the fields defined in example 1 contain the following:

A	0200	0201	0202
(A)	0100 1110	0100 0011	0101 0010

B	0300	0301	0302
(B)	0100 1001	0100 0010	0100 1101

Binary Comparison:

```
1's Complement of (A)    10110001    10111100    10101101
                  (B)    01001001    01000010    01001101
                         11111000    11111110    11100000    Partial sum
                                1                     11 1    Carries
                                                         1    Initial carry
                         11111010    11111110    11111011    Final sum
```

Conditions:

1. Number unequal indicator ON (final sum ≠ 0).

2. No carry beyond the specified field. The G flag is set ON and the repeat indicator is turned OFF, if it is ON.

C = 102(66)(E6)

The REPEAT command sets up conditions that cause the command immediately following it to be repeated n times (repeat number) or skipped (if n = 0). The repeat number is the contents of a 1-character field specified by the effective A address, and may range in value from 0 through 255, with 0 = 0 in this case. Any command may follow a REPEAT command, but certain commands nullify repeat preparation.

The REPEAT command moves the repeat number into a counter. If the repeat number is not equal to zero, the repeat indicator is turned ON and the command terminates. If the repeat number is equal to zero, the repeat indicator is not turned on, and the control register is incremented to access the second command following the REPEAT command.

The T and B portions of the command are not used. If a 2-address command is specified, the new T and B values are stored in the appropriate registers.

Near the completion of the setup phase for each command, the repeat indicator is tested. If a PACK, UNPACK, ADD, SUBTRACT, MVAR, or COMPARE command is being set up at the time, and the repeat indicator is ON, the contents of the control register are left unchanged so that the command being repeated will be referenced and the repeating flow is entered.

In the repeating flow, the repeat counter is decremented by 1 and compared to 0. If the result is not equal, the command being repeated is performed. If the result is equal, the repeat indicator is turned OFF, the control register is incremented to contain the address of the next command, and the command being repeated is performed for the last time. When the repeating flow is entered, the contents of the repeat counter are compared to 0 before decrementation. If the result is equality, a program error (PE) is indicated.

If a BRANCH, WAIT, or INOUT command follows a REPEAT command and the repeat number is greater than 0, the repeat indicator is turned OFF, nullifying the REPEAT command. If the repeat number is equal to 0, the command following is skipped.

If a REPEAT command is followed by a second REPEAT command and the repeat number of the first command is not equal to 0, the first command is nullified and the second is performed. If the repeat number of the first command is equal to 0, the second REPEAT command is skipped.

Command Execution Time

    E = PO + 2P if the value in the REPEAT COUNTER $\neq$ 0
    E = PO + 3P if the value in the REPEAT COUNTER = 0

C = 103(67)(E7)

The WAIT command causes program operation to stop functionally. Program interruption as well as input and output operations can occur.

The repeat indicator is turned OFF, and the WAIT indicator on the operator's console is lighted. The 16 bits of A are also displayed (19 bits in systems having the extended memory option).

The compute indicator, which is set on by pressing the COMPUTE switch on the console is tested. If it is ON, it is turned OFF; the WAIT indicator is turned OFF, and the next command in sequence is accessed. If the compute indicator is OFF, the control register is decremented by the size of the WAIT command and the command terminates, permitting between commands testing.

B and T are not used.

Command Execution Time

E = PO + 2P

NOTE

The WAIT command is reexecuted (including setup) unless the COMPUTE button is pressed.

Command		Code
BRANCH OVERFLOW	(BROV)	104(68)(E8)
BRANCH LESS	(BRL)	105(69)(E9)
BRANCH EQUAL	(BRE)	106(6A)(EA)
BRANCH LESS OR EQUAL	(BRLE)	107(6B)(EB)
BRANCH GREATER	(BRG)	108(6C)(EC)
BRANCH LESS OR GREATER	(BRU)	109(6D)(ED)
BRANCH GREATER OR EQUAL	(BRGE)	110(6E)(EE)
BRANCH UNCONDITIONALLY	(BR)	111(6F)(EF)

Each branch command (except BRANCH UNCONDITIONALLY) tests its appropriate flag (G, L, E, or OF) and the repeat indicator is turned OFF. If the flag is OFF, the next command in sequence is performed. The BRANCH UNCONDITIONALLY command takes the branch in all cases.

If the appropriate flag is ON, or a BRANCH UNCONDITIONALLY command is specified, the next command performed is that indicated by the effective A address. If this is not a legal command address, a PE occurs and the control register is undisturbed.

When the overflow branch is taken, the OF flag is turned OFF.

The T and B portions of any branch command are not used. If a 2-address command is specified, the T and B values are stored in the appropriate registers where they are available for subsequent implied T and B operations.

Command Execution Time

    E = PO + P if branch unsuccessful.
    E = PO + 2P if branch successful.
    E = PO + P if branch unconditional.

C = 112(70)(FO)

The effective A address is the address of the peripheral address field (PAF).

The INOUT command initiates the specified I/O functions by transmitting the PAF, via the appropriate trunk, to the peripheral, which selects itself according to the PAF contents.

After the processor has transmitted the last PAF character, the appropriate S2 status character is stored and the command terminates. Certain status conditions cause the command to terminate before the PAF is exhausted, and an S2 status character is stored according to the condition that caused the command termination. If the command terminates for any reason other than a selection, the peripheral is not activated and is not in a select state at the completion of the command.

The effective B address is the location into which the S2 status character is stored. A location different from the status character location in the control word can be designated so that both S2 and S3 status characters may be preserved if desired.

T is used only in those systems with the multiprogramming option.

Command Execution Time
─────────────────────

For the integrated printer:

   E = 2PO + 6P

For other INOUT operations:

   E = PO + 4P + L(2PO + 7P + M + N)

   L = Number of characters in PAF
   M = Response time for character received signal = 0 to 35000 nanoseconds
   N = Response time if character received signal is not sent. $N > 0$
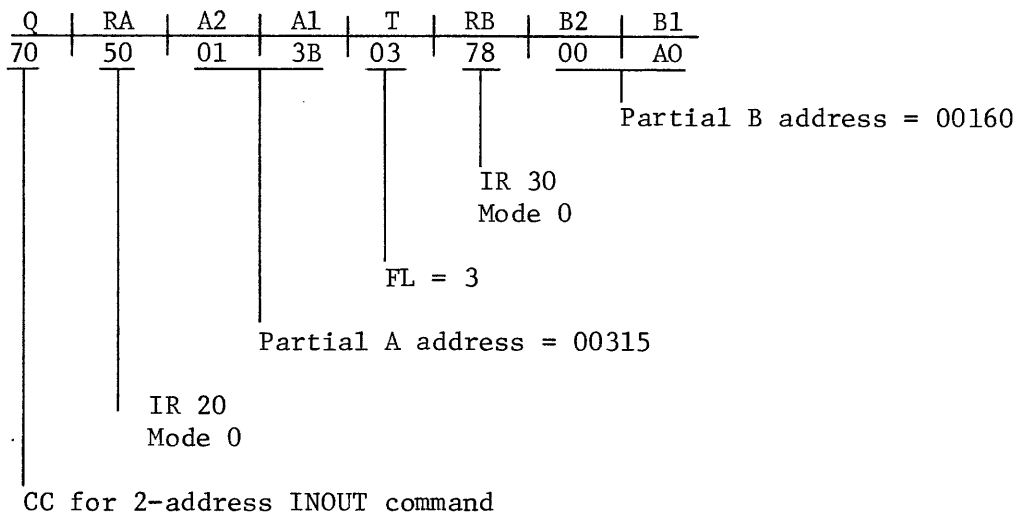
```
  Q  |  RA  |  A2  |  A1  |  T  |  RB  |  B2  |  B1
 70  |  50  |  01  |  3B  | 03  |  78  |  00  |  AO
```

Partial B address = 00160

IR 30
Mode 0

FL = 3

Partial A address = 00315

IR 20
Mode 0

CC for 2-address INOUT command

Assume the index registers contain the following:

    (IR20)   07D0   (02000)
    (IR30)   0BB8   (03000)

After command setup:

Effective A address = 07D0   013B   090B   (02315)
Effective B address = 0BB8   00A0   0C58   (03160)

Before command execution:

```
 A  |   090B    |   090C    |   090F
(A) | 0000 0001 | 0001 000  | 001 0010

 B  |   0C58
(B) | 0001 0001
```

A is the address of the first
character of a PAF, which will
vary in length according to the
peripheral desired.

Following command execution:

B [0C58]   B is the memory address containing the S2 character
(B) [0000 0000]   according to the result of the selection attempt.

A is unchanged.

T is used only with the multiprogramming option.

Steps involved in the execution of the INOUT command are:

1. The PAF (beginning at address 090B in the above example) is accessed
   and the selection process begins. If either the trunk or the unit is
   busy or the unit is inoperable a status character is stored in memory
   (address 0C58 in above example) and the command terminates.

2. If neither the trunk nor the unit is busy and the unit is operable, the
   remaining portion of the PAF is transmitted to the peripheral unit, one
   character at a time.

3. When the peripheral signals that no more PAF characters are required,
   the status character is stored and the command terminates. In the above
   example, S2 is stored in address 0C58.

4. If the required peripheral responses are not received for any reason,
   the command terminates and an S2 is stored; the peripheral is not
   normally activated nor left in a selected state at the completion of
   the command.

For a detailed description of peripheral operation with the INOUT command,
refer to the I/O Control section of the 200 processor publication.

INTRODUCTION

The 1401 simulator option consists of five commands designed to reduce the time and space requirements for the software simulation of 1401 programs.

Systems with the 1401 simulator option always contain the LOGIC and TABLE COMPARE commands. These commands may also be included in systems that do not have the option.

COMMAND OPERATION

The following conventions apply to all 1401 simulator commands with the exception of the 1401 CONVERT command:

- The effective A or B address references the rightmost character of the respective A or B operand. All operands are processed from right to left.
- The leftmost character of each operand is the first character encountered that has b8 equal to 1.
- Bits b8 and b7 of every operand character remain undisturbed regardless of any changes to the remaining bits of the character.
- A negative sign in a numeric field is indicated by b6 ON and b5 OFF in the rightmost character position of the field. Any other combination is considered to be a plus sign. When the processor sets the sign to plus, as it does in performing a complement add, b6 and b5 are both set ON.

The decimal values are represented by the following bit configurations:

DECIMAL VALUE	B4	B3	B2	B1
0	1	0	1	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

Four zero bits (0000) are also recognized and treated as decimal zero.

The appearance of the remaining hexadecimal bit configurations in either operand may produce results that are incompatible with the 1401.

$$C = 56(38)(B8)$$

The contents of the field specified by A are algebraically added to the contents of the field specified by B, and the result replaces the B field. The A field remains undisturbed.

Bits 5 and 6 of the rightmost character of each field contain the algebraic sign of the field.

- Like Signs

  When adding numbers with like signs, bits 5 and 6 of all characters except the rightmost and leftmost characters of the B field are cleared to zero.

  Bits 5 and 6 of the leftmost character have a 1 added to them each time an overflow occurs and the overflow flag (OF) is set ON.

- Unlike Signs

  When adding numbers with unlike signs, bits 5 and 6 of all characters except the rightmost character of the B field are cleared to zero.

  The result of adding two numbers with unlike signs is stored in its true form.

The command terminates when the leftmost character of B is encountered. If the A field is shorter than the B field, it is treated as though it contained a sufficient number of high order zeros to make its length equal to the length of B. If the A field is longer than the B field, only the low-order characters of A, as determined by the length of B, are processed.

The initial B address is stored in the 18 low-order bits of a 4-character field at location 00348.

If (A) and (B) overlap, the result may differ from the result of an operation where the fields do not overlap.

The T portion of the command is not used.

The implied B, after command execution, is equal to the address of the character to the left of the last B character referenced. The B portion of the command is not affected.

$$(B) = (B) + (A)$$
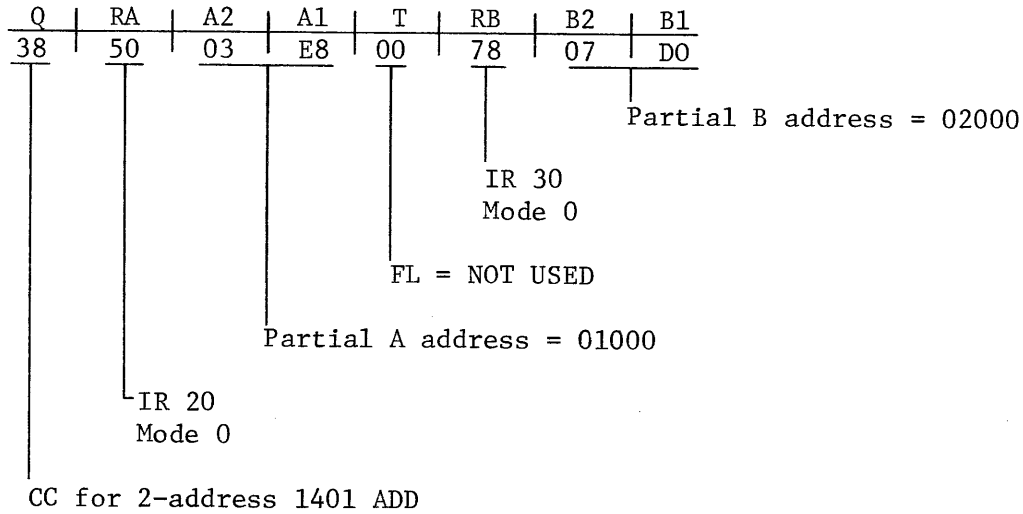
EXAMPLE   1401 ADD

```
 Q  | RA | A2 | A1 | T  | RB | B2 | B1
 38 | 50 | 03 | E8 | 00 | 78 | 07 | D0
```

Partial B address = 02000

IR 30
Mode 0

FL = NOT USED

Partial A address = 01000

IR 20
Mode 0

CC for 2-address 1401 ADD

Assume the index registers contain the following:

    (IR20) = 2710 (10000)
    (IR30) = 3A98 (15000)

After command setup:

Effective A address = 2710 + 03E8 = 2AF8 (11000)
Effective B address = 3A98 + 07D0 = 4268 (17000)

Before command execution:

A	2AF4	2AF5	2AF6	2AF7	2AF8
(A)	0100 0101	0000 0100	0000 1000	0000 0111	0111 0011

B	4264	4265	4266	4267	4268
(B)	0100 0110	1000 1000	0100 1010	0100 1001	0111 0101

In location 4265 of the B field, b8 ON specifies the leftmost character.

Positive numbers are specified by b6 and b5 of the rightmost characters of the A and B fields.

Addition of the two fields is performed. (A)   4873
                                          (B) + 8095
                                                12968

The result replaces the contents of B.

B	4264	4265	4266	4267	4268
(B)	0100 0110	1001 0010	0100 1001	0100 0110	0111 1000

The carryout of the thousands position is stored in b5 of location 4265 of the B field.

The initial B address is stored in the 18 low-order bits of a 4-character field at location 015C (00348).

015C	015D	015E	015F
		42	68

$\underline{B}$ = 4264 (hexadecimal address of the character to the left of the last B character referenced).

1401 SUBTRACT

C = 57(39)(B9)

This command functions in the same manner as the 1401 ADD except that the field specified by A is algebraically subtracted from the field specified by B, and the result replaces the B field.

$\underline{(B)}$ = (B) − (A)

1401 SCAN

C = 58(3A) (BA)

The contents of the field specified by A are scanned to locate the leftmost character. The address of the character to the left of the leftmost character is stored into IR9.

The T and B portions of the command are not used.

EXAMPLE   1401 SCAN

Q	RA	A2	A1	T	RB	B2	B1
3A	64	01	F4	00	00	00	00

Partial B address = NOT USED

IR NOT USED
Mode NOT USED

FL = NOT USED

Partial A address = 00500

└IR25
Mode 0

CC for 2-address 1401 SCAN
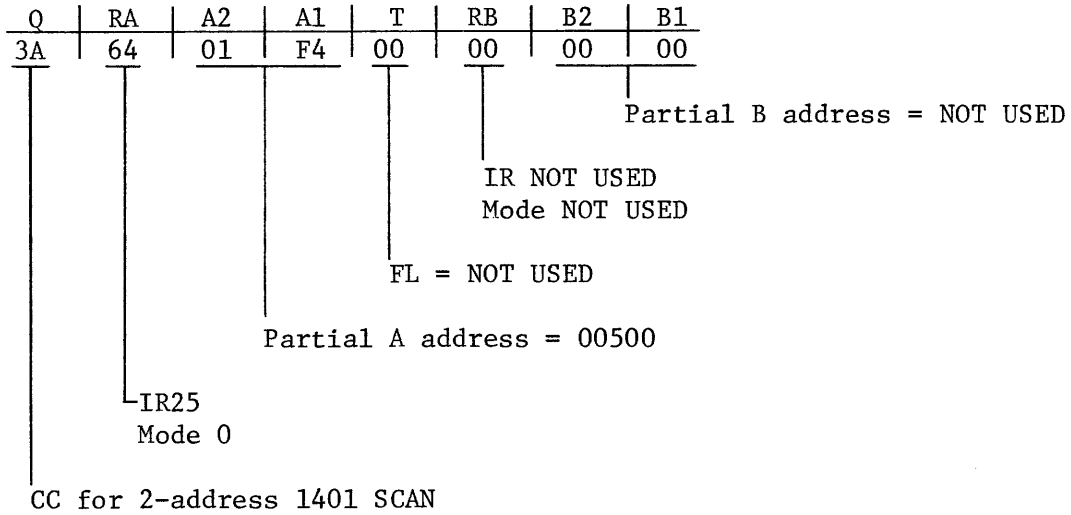
Assume the index registers contain the following:

(IR 25) = 07D0 (02000)

After command setup:

Effective A address = 07D0 + 01F4 = 09C4 (02500)
Effective B address = NOT USED

The contents of A remain unchanged.

A	09BE	09BF	09C0	09C1	09C2	09C3	09C4
(A)	0011 0111	1111 0110	0111 0101	0011 0100	0111 0011	0011 0010	0011 0001

Index Register	0024	0025	0026	0027
Word 9			09	BE

C = 59(3B)(BB)

Each character in the field specified by A is moved into the corresponding location of the field specified by B, one character at a time, starting with the rightmost character. The two high order bits (b8 and b7) of each character are undisturbed.

The command terminates when the leftmost character is encountered and processed in either the A or B field.

The T portion of the command is not used.

The implied B after command execution is equal to the address of the character to the left of the last B character referenced. The B portion of the command is not affected.

(B) = (A)
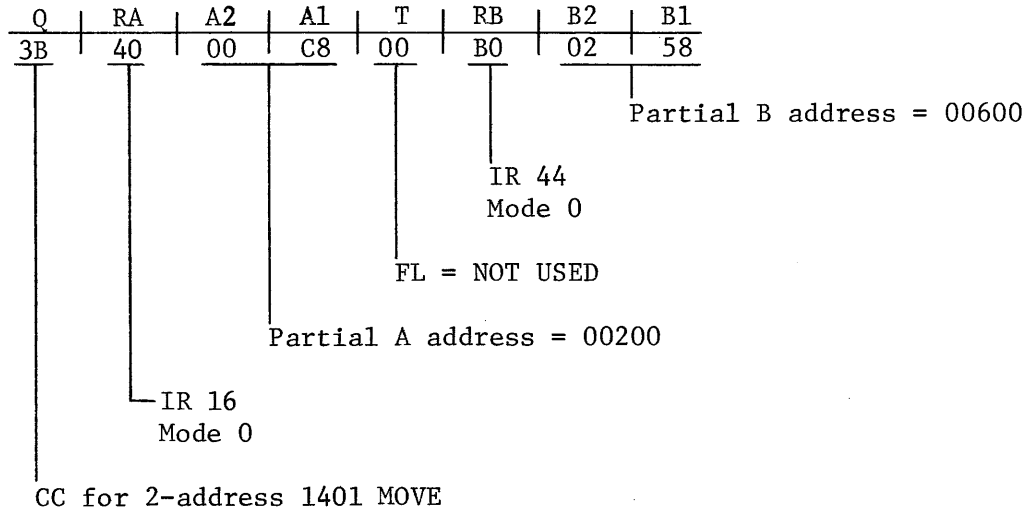
EXAMPLE   1401 MOVE

```
   Q  | RA  | A2  | A1  |  T  | RB  | B2  | B1
   3B | 40  | 00  | C8  | 00  | B0  | 02  | 58
```

Partial B address = 00600

IR 44
Mode 0

FL = NOT USED

Partial A address = 00200

IR 16
Mode 0

CC for 2-address 1401 MOVE

Assume the index registers contain the following:

    (IR 16) = 0640 (01600)
    (IR 44) = 1130 (04400)

After command setup:

Effective A address = 0640 + 00C8 = 0708 (01800)
Effective B address = 1130 + 0258 = 1388 (05000)

Before command execution:

A	0702	0703	0704	0705	0706	0707	0708
(A)	0001 1001	1001 0110	0011 0001	0011 0111	0010 0101	0111 0101	0010 1001

B	1382	1383	1384	1385	1386	1387	1388
(B)	0011 0001	0010 1001	0010 0110	0011 0010	0111 0101	0010 1001	0001 0011

The leftmost character is specified by b8 equal to 1 in location 0703 of the
A field.

The contents of A replace the contents of B, including the leftmost character.

After command execution:

B	1382	1383	1384	1385	1386	1387	1388
(B)	0011 0001	1001 0110	0011 0001	0011 0111	0010 0101	0111 0101	0010 1001

B = 1382

C = 60(3C)(BC)

The 4-character field specified by A contains an address in 1401 form in the three rightmost character positions. The four rightmost bit positions of each character represent addresses from 000-999. Bits 5 and 6 of the units and hundreds positions are used to indicate addresses from 1,000 to 15,000. Combinations of these bit positions may indicate a number up to 15,999. Bits 7 and 8 of all three characters are ignored by the command, as are bits 5 and 6 of the tens (middle) character.

The converted 1401 address is stored in the 4-character field specified by B. The 1401 address in A is converted to its binary equivalent and added to a number called the base. The sum is stored in the 18 low order bits of B. The base is located in the 16 low order bits of the 4-character field at address 00344 (low order 19 bits when the extended memory option is included).

The fields specified by A and B must be at locations that are evenly divisible by 4 or a PE results.

The T portion of the command is not used.

B = B

NOTE

A bit configuration of 1010 or 0000 is recognized as a decimal zero in the four rightmost bit positions of each character in the 1401 address (field specified by A).
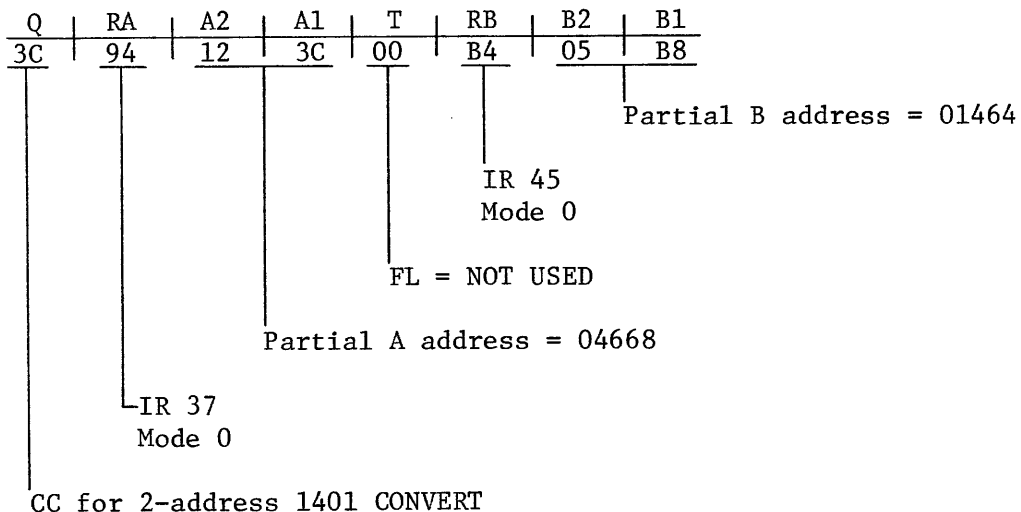
EXAMPLE   1401 CONVERT

```
 Q  |  RA  |  A2  |  A1  |  T  |  RB  |  B2  |  B1
 3C |  94  |  12  |  3C  |  00 |  B4  |  05  |  B8
```

Partial B address = 01464

IR 45
Mode 0

FL = NOT USED

Partial A address = 04668

IR 37
Mode 0

CC for 2-address 1401 CONVERT

Assume the index registers contain the following:

(IR 37) = 0E74 (03700)
(IR 45) = 1194 (04500)

After command setup:

Effective A address = 0E74 + 123C = 20B0 (08368)
Effective B address = 1194 + 05B8 = 174C (05964)

The contents of A remain unchanged.

	HUNDREDS	TENS	UNITS
	20B0	20B1	20B2	20B3
A				
(A)	0000 0000	0000 0111	0000 1000	0000 1001

NOT USED   IGNORED

   1401 address = 00789

Convert 1401 address to its binary equivalent.

        1401 FORM   BINARY
            00789 = 0000 0011 0001 0101

Base number (location 00344 = 0158 hex).

0158	0159	015A	015B
0000 0000	0000 0000	0000 0011	1110 1000

Add converted 1401 address to the base number.

```
    0000 0000  0000 0000  0000 0011  1110 1000
+                         0000 0011  0001 0101
    _____
    0000 0000  0000 0000  0000 0110  1111 1101
```

Store sum in B.

B
(B)

20B0	20B1	20B2	20B3
0000 0000	0000 0000	0000 0110	1111 1101

INTRODUCTION

The 315 simulator option consists of three additional NCR Century 200 commands (EXECUTE, PACK-SPECIAL, UNPACK-SPECIAL) and a freestanding unit that accesses the NCR Century memory and executes 315 commands. The freestanding unit also includes two addressing registers, the base address register (BAR) for relocating the 315 program in the NCR Century memory, and the limit address register (LAR) for controlling the program's addressing range. The freestanding unit does not operate with addresses above 131,071 (17 bits); therefore, in systems having maximum memory size, the freestanding unit is attached to the low-order 128K only. This option requires reserved NCR Century memory area at locations 6144-6355 (hex 1800-18D3).

DATA REPRESENTATION

The 315 simulator option considers a character stored in the NCR Century memory to be in 6-bit, 315 format. Only the six low-order bits of each character are used to contain information. Each pair of characters (two adjacent NCR Century memory locations) is equivalent to a 315 slab.

NCR CENTURY/315 MEMORY FORMAT

NCR Century format	HT*	r
315 format	9	T
Binary (NCR Century)	0 0 0 0 1 0 0 1	0 1 1 1 0 0 1 1

Binary (315)   0 0 1 0 0 1 1 1 0 0 1 1

315 slab (Digit)	2	7	3

(Alpha)	9	T

* Special Characters. See NCR Century Code Chart.
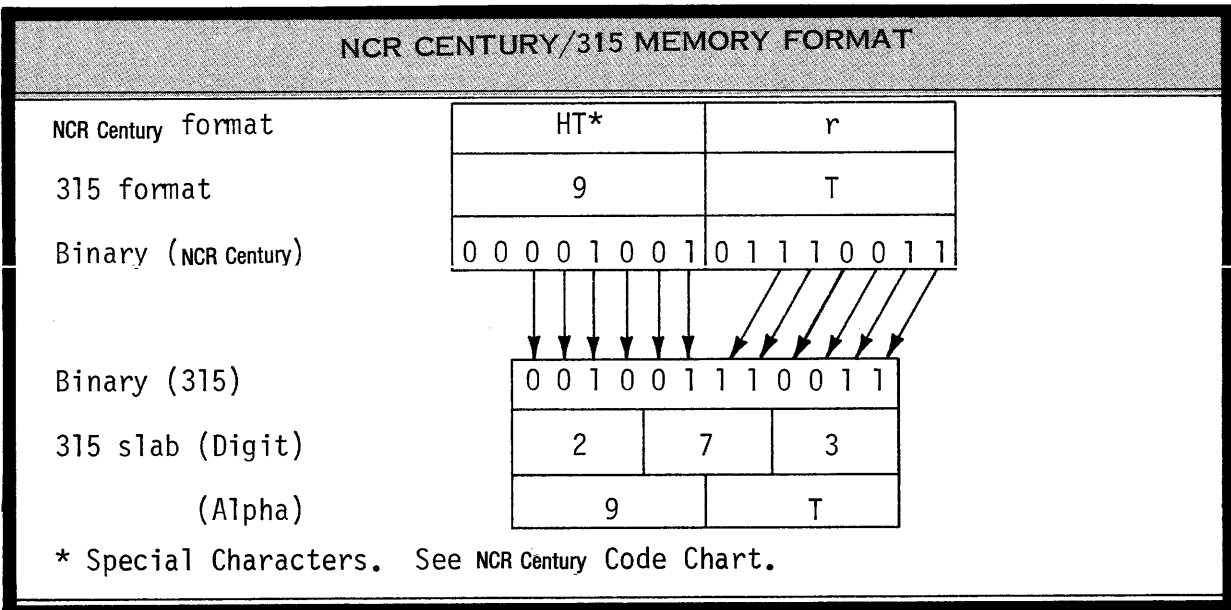
The following table indicates the 315 commands that the freestanding unit can execute. Any command not on this table is interpreted by software after the freestanding unit performs a trap operation.

315 COMMANDS EXECUTED BY THE PROCESSOR

Code	F	C	Q	G	Code	F	C	Q	G
LD		1			CNT	1	0		
ST		2			LD:R	2	0	0	0
ADD		3			LD:J	2	0	0	1
SUB		4			SLD:R	2	0	0	2
COMP		6			SLD:J	2	0	0	3
* TEST: G	0	7			MOVE: RR	2	0		4
* TEST: L	1	7			MOVE: JR	2	0		5
* TEST: E	2	7			MOVE: RJ	2	0		6
* JUMP	5	7			MOVE: JJ	2	0		7
ADD: M		9			ST: R	2	0		8
TEST: LH	0	∅			ST:J	2	0		9
TEST: RH	1	∅			AUG: R	2	0	1	0
SET F: LH	2	∅			AUG: J	2	0	1	1
SET F: RH	3	∅			SAUG: R	2	0	1	2
CLRF: LH	4	∅			SAUG: J	2	0	1	3
CLRF: RH	5	∅			MOVE: B	3	0	0	
					MOVE: E	3	0	1	

* R ≠ 15.

NOTE

In the COUNT, MODIFY (F=2, C=0), and MOVE commands, no check is made for illegal modes, i.e., the command will be executed as one of the legal modes if an undefined mode is specified. When data is transferred between the index (or jump) registers and the memory, the data is automatically converted to its proper form. Also, the simulated control register (IR 31) in memory is updated before any mode of the MODIFY command is executed. A decimal to binary conversion occurs when data is transferred from memory to index or jump registers.

315 RMC programs containing unique RMC modes of these commands will not be executed properly. 315 programs which have been written with violations of command modes or augmenters will not be executed properly.

RESERVED MEMORY

Within the NCR Century 200 memory is an area reserved by the 315 simulator option. This area occupies locations 6144 – 6355 (hex. 1800-18D3) and is divided into two main areas; 315 simulated index registers and jump registers and 315 command trapping information.

Simulated Index and Jump Registers

The 315 auxiliary memory is simulated in fixed locations within the NCR Century 200 memory. Each 315 index and jump register is represented by two characters in the NCR Century 200 memory. The information contained in these registers must be in binary form, using all eight bits of each pair of characters. Register locations begin at 6144 (hex 1800) and extend through 6271 (hex 187F), with IR 0, Group 0 at 6144 (hex 1800), IR 1, Group 0 at 6146 (hex 1802); refer to Table 1 for all register locations.

315 Command Trapping Information

This area contains the new control address which will be transferred to the NCR Century 200 upon detection of a 315 command that cannot be executed by the freestanding unit. The command code of the 315 instruction determines which address is stored as the new control address (see table 2). The freestanding unit also stores certain information pertaining to the 315 command (see table 3) and updates the 315 sequence control register (IR31, location 6206 and 6207) in the NCR Century memory.

This area also contains the trap control register, the 315 simulated accumulator, flags, and the BAR-LAR registers.

Option Switches

The 315 option switches 0-7 correspond to NCR Century 200 option switches 1-8. A template may be placed over the switches on the NCR Century Console for operator convenience.

TABLE 1 – SIMULATED INDEX AND JUMP REGISTERS

DECIMAL	HEXADECIMAL	INDEX REGISTER GROUP O	RELATIVE	DECIMAL	HEXADECIMAL	JUMP REGISTER GROUP O	RELATIVE
6144-6145	1800-1801	0	0	6208-6209	1840-1841	0	0
6146-6147	1802-1803	1	1	6210-6211	1842-1843	1	1
6148-6149	1804-1805	2	2	6212-6213	1844-1845	2	2
6150-6151	1806-1807	3	3	6214-6215	1846-1847	3	3
6152-6153	1808-1809	4	4	6216-6217	1848-1849	4	4
6154-6155	180A-180B	5	5	6218-6219	184A-184B	5	5
6156-6157	180C-180D	6	6	6220-6221	184C-184D	6	6
6158-6159	180E-180F	7	7	6222-6223	184E-184F	7	7
6160-6161	1810-1811	8	8	6224-6225	1850-1851	8	8
6162-6163	1812-1813	9	9	6226-6227	1852-1853	9	9
6164-6165	1814-1815	10	10	6228-6229	1854-1855	10	10
6166-6167	1816-1817	11	11	6230-6231	1856-1857	11	11
6168-6169	1818-1819	12	12	6232-6233	1858-1859	12	12
6170-6171	181A-181B	13	13	6234-6235	185A-185B	13	13
6172-6173	181C-181D	14	14	6236-6237	185C-185D	14	14
6174-6175	181E-181F	15	15	6238-6239	185E-185F	15	15
		GROUP 1				GROUP 1	
6176-6177	1820-1821	0	16	6240-6241	1860-1861	0	16
6178-6179	1822-1823	1	17	6242-6243	1862-1863	1	17
6180-6181	1824-1825	2	18	6244-6245	1864-1865	2	18
6182-6183	1826-1827	3	19	6246-6247	1866-1867	3	19
6184-6185	1828-1829	4	20	6248-6249	1868-1869	4	20
6186-6187	182A-182B	5	21	6250-6251	186A-186B	5	21
6188-6189	182C-182D	6	22	6252-6253	186C-186D	6	22
6190-6191	182E-182F	7	23	6254-6255	186E-186F	7	23
6192-6193	1830-1831	8	24	6256-6257	1870-1871	8	24
6194-6195	1832-1833	9	25	6258-6259	1872-1873	9	25
6196-6197	1834-1835	10	26	6260-6261	1874-1875	10	26
6198-6199	1836-1837	11	27	6262-6263	1876-1877	11	27
6200-6201	1838-1839	12	28	6264-6265	1878-1879	12	28
6202-6203	183A-183B	13	29	6266-6267	187A-187B	13	29
6204-6205	183C-183D	14	30	6268-6269	187C-187D	14	30
6206-6207	183E-183F	15	31	6270-6271	187E-187F	15	31

Location of new control addresses when a trap occurs on a 315 command. Each command code that traps can be made to transfer control to a different location.

TABLE 2 — TRAP LOCATIONS

LOCATION *		COMMAND	F	C
DECIMAL	HEXADECIMAL			
6272-6281	1880-1889	(See table 4-information about the command causing the trap)		
6282-6283	188A-188B	MULTIPLY		5
6284-6285	188C-188D	(See table 4-information about the command causing the trap)		(6)
6286-6287	188E-188F	TEST		7
6288-6289	1890-1891	SHIFT		8
6290-6291	1892-1893	LAR violation		9
6292-6293	1894-1895	ADD BINARY		@
6294-6295	1896-1897	DIVIDE		,
6296-6297	1898-1899	JUMP		
6298-6299	189A-189B	EDIT		&
6300-6301	189C-189D	SUPPRESS		.
6302-6303	189E-189F	Illegal command	0	0
6304-6309	18A0-18A5	Used by the hardware as temporary storage		
6310-6311	18A6-18A7	SCAN	4	0
6312-6313	18A8-18A9	PACK	5	0
6314-6315	18AA-18AB	PUTAWAY	6	0
6316-6317	18AC-18AD	INTERROGATE	7	0
6318-6319	18AE-18AF	Illegal command	0	-
6320-6321	18B0-18B1	PLACE	1	-
6322-6323	18B2-18B3	READER	2	-
6324-6325	18B4-18B5	BUFFER	3	-
6326-6327	18B6-18B7	MAGNETIC TAPE	4	-
6328-6329	18B8-18B9	MAGNETIC CARD (CRAM)	5	-
6330-6331	18BA-18BB	INQUIRY	6	-
6332-6333	18BC-18BD	Illegal command	7	-
6334-6335	18BE-18BF	TRAP CONTROL REGISTER		
6336-6351	18C0-18CF	ACCUMULATOR		
6352-6353	18D0-18D1	Flags (See Table 3)		
6354-6355	18D2-18D3	BAR-LAR		

*The locations given contain the addresses of entry points to software sub-routines that simulate the 315 command.

TABLE 3 — FLAG AND INDICATOR LOCATIONS		
LOCATION	BITS	USE
6352	B8 - B5	Not used
	B4	Less than flag
	B3	Equal flag
	B2	Greater than flag
	B1	Overflow flag
6353	B8 - B7	Not used
	B6	K@; sign of accumulator
	B5 - B1	Length of accumulator in characters

BASE AND LIMIT ADDRESS REGISTERS

Two registers located in the freestanding unit are used for addressing the NCR Century 200 memory, the base address register (BAR) and the limit address register (LAR). The contents of both these 6-bit registers are used as bit positions 17 through 12 in address calculations. This yields an addressing range of 2048 to 131,071.

Base Address Registers

The contents of the base address register are used to offset the contents of IR31 to determine the location on a 315 command.

Bit	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
(IR31)		X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
(BAR)	X	X	X	X	X	X											
Location	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

When the freestanding unit is executing a command and accessing the NCR Century 200 memory, the contents of BAR form the base address for all memory accesses except those made to the fixed, reserved locations 6144-6355. The contents of the BAR are added to bit positions 17 through 12 of the relative address prior to the memory access.

Limit Address Register

The contents of LAR specify the upper limit for the memory access by the freestanding unit. If the contents of LAR are greater than bits 17 through 12 of the relative address (prior to adding the contents of BAR), the memory access is permitted. If the contents of LAR are not greater than bits 17 through 12 of the relative address, the memory access is not permitted, and control is returned to the NCR Century 200 at the address found in reserved memory location 6290-6291 (hex 1892-1893). The address found in the location 6290-6291 (hex 1892-1893) is moved to the trap control register at location 6334-6335 (hex 18BE-18BF), and is subsequently loaded into the NCR Century 200 control register.

Program Interrupt

The 315 program may be interrupted by the same conditions that cause interrupt in the NCR Century 200 processor: I/O termination when the interrupt permit indicator is ON, and the trace permit being ON in the systems that have the trace option.

Interrupt occurs at the completion of the command being executed or immediately after control is returned to the NCR Century 200. When interrupt occurs at the completion of command execution, the control register stored in the program status word references the location of the EXECUTE command. When interrupt occurs immediately after control is returned to the NCR Century 200, the control register stored in the program status word references the new control location (the location to which control would have been transferred had interrupt not occurred). In most cases this is the next command in sequence.

FUNCTIONAL OPERATION

The freestanding unit is instructed to operate when the NCR Century 200 encounters an EXECUTE command. The EXECUTE command causes the NCR Century 200 to loop within the execute flows and to give control to the freestanding unit. The freestanding unit loads its base address and limit address registers from NCR Century memory locations 6354-6355 (hex 18D2-18D3). To determine the location of the 315 command, the freestanding unit adds the contents of BAR and the contents of IR31 (location 6206-6207).

The freestanding unit reads the memory location calculated, analyzes the contents to determine what the 315 command is and checks to see if the command is one which it can execute. If the command can be executed, the freestanding unit executes it, counts IR31 up by the command length and is ready to access the next command in sequence.

If the freestanding unit reads out the new command and determines that the command cannot be executed, the unit loads a specific address into the trap control register located in NCR Century memory locations 6334-6335 (hex 18BE-18BF). The address stored in the trap control register is determined by the 315 command code which is decoded into a specific address for a specific command. Therefore, it is possible to specify where the program should go depending upon the particular command which cannot be executed by the hardware.

The freestanding unit also stores pertinent data related to the 315 command when trapping (see table 3). This data is stored in locations 6272 through 6285 (hex 1880-188D) with the exception of 6282-6283 (hex 188A-188B), which is unchanged.

The freestanding unit then increments the contents of IR31 by the length of the 315 command that caused the trapping and turns control over to the NCR Century 200. The NCR Century 200 exits the loop and reads out the contents of the trap control register. The contents are the trap address which contains the address of the particular subroutine necessary for software interpretation of the 315 command. The EXECUTE command terminates with the sequence control register set to the address of the appropriate routine to be entered. The NCR Century 200 enters the between commands testing (BCT) flow. Since no BCT trap conditions exist, the next command in sequence is initiated using the address just stored in the sequence control register, which is the address of the subroutine desired.

<u>NOTE</u>

Upon completion of the subroutine the programmer should either do a direct jump to an EXECUTE command or have an EXECUTE command as the last command for every subroutine to which the 315 program may trap.

Each time the freestanding unit uses the NCR Century 200 memory, a check is made prior to memory access to see if the calculated address (without the contents of BAR) is greater than the contents of LAR. If the address is larger than (LAR), memory access is not permitted and the LAR violation causes a trapping operation. The freestanding unit stores related 315 information in locations 6272 through 6281 (hex 1880-1889) and 6282-6283 (hex 188C-188C). The address of the LAR violation control register is stored in the trap control register which is then transferred to the sequence control register and control of the operation is given to the NCR Century 200. The NCR Century 200 enters BCT and upon exiting uses the sequence control register for the next command in sequence (the address of the subroutine for processing the LAR violation).

TABLE 4—INFORMATION STORAGE FOR A TRAPPED 315 COMMAND		
LOCATION DECIMAL	HEX.	INFORMATION
6272-6275	1880-1883	2(RxA) + (BAR) = AbsoluteNCR Century 200 address of the 315 operand in memory when R≠15 OR A address + (BAR) when R = 15

6276	1884	KS flag bit 1=1 if literal,
		           0 if nonliteral
		
		bit 2=1 if single,
		           0 if double stage

6277	1885	*2F + 2 = Field length in NCR Century characters
6278-6279	1886-1887	**JY address-absolute address of the register specified by the JY of the 315 command when JY refers to an index register; if JY refers to a jump register, a binary 64 must be added to the address by the program.
6280	1888	**Q portion of the 315 command
6281	1889	**G portion of the 315 command
6282-6283	188A-188B	undisturbed
6284-6285	188C-188D	**B portion of the 315 command stored in binary

* F = 315 F, 315 F = total number of slabs in field minus 1.

** = Double stage commands only

If a LAR violation is caused by the 315 control register (IR31) contents exceeding the LAR contents, b2 is set to 1 even though no command size is involved. In this case, IR31 contains the illegal address + 1. When a LAR violation occurs during the execution phase of a 315 command, IR31 contains the address of the next command to be executed.

The LAR check is made on the address specified by 2(RxA) before this address is stored. If a LAR violation·occurs, control is transferred to the NCR Century 200; the NCR Century· 200 sequence control register has already been loaded from the trap control register, which contains the address of the subroutine. The address is retained in the LAR violation register located at 6290-6291.

Errors

When a memory error is detected by the freestanding unit, the operation terminates and control is returned to the NCR Century 200. The NCR Century enters the memory error trapping flow and stores the location of the EXECUTE command as the control register in the error status word.

When an illegal address is detected by the freestanding unit it is treated as a LAR violation and control is returned to the NCR Century 200 at the address specified for that purpose. No other program errors are detected by the freestanding unit.

C = 61(3D)(BD)

This command causes the NCR Century 200 to loop within the flows and give control to the freestanding unit. The freestanding unit will execute 315 commands until it encounters a command that it cannot execute, a LAR violation, a program interrupt or a memory error. When one of these conditions is detected, the freestanding unit performs a trapping operation that stores related data and loads the NCR Century 200 sequence control register with the address of a subroutine to process the condition. When control is returned to the NCR Century 200 because of a command code trap or a LAR violation the new control location for the NCR Century is taken from the trap control register, location 6334-6335.

When control is returned to the NCR Century 200 because of a program interrupt, the new control location is determined by the between commands testing.

When the freestanding unit detects a memory error, the operation terminates and control is returned to the NCR Century 200 which turns on its ME indicator and enters the normal memory error trapping flow as if it had detected the ME. While in this flow, the NCR Century 200 will store the sequence control register which at this time contains the address of the EXECUTE command.

This command may not be repeated and resets the repeat indicator. If the multiprogramming option is installed, EXECUTE will be a priviledged command.

$$C = 62(3E)(BE)$$

(A) is packed and moved to (B).

The four low-order bits of every three characters of (A) are moved into the six low-order bits of two characters of (B), starting with the leftmost character of each field. The two high-order bits of each character of (B) are set to 0.

The beginning address of the B field must be evenly divisible by 2 or a program error occurs.

T specifies the length of (A) in triplets and ranges from 0 through 255, with 0 = 256. (T=1 for each 3 characters of the A operand.)

$\underline{B}$ = B + 2T, except when T = 0, in which case B = B + 512.

<u>NOTE</u>

Special PACK and UNPACK commands are required in NCR Century hardware since the 315 simulation unit cannot execute these commands.
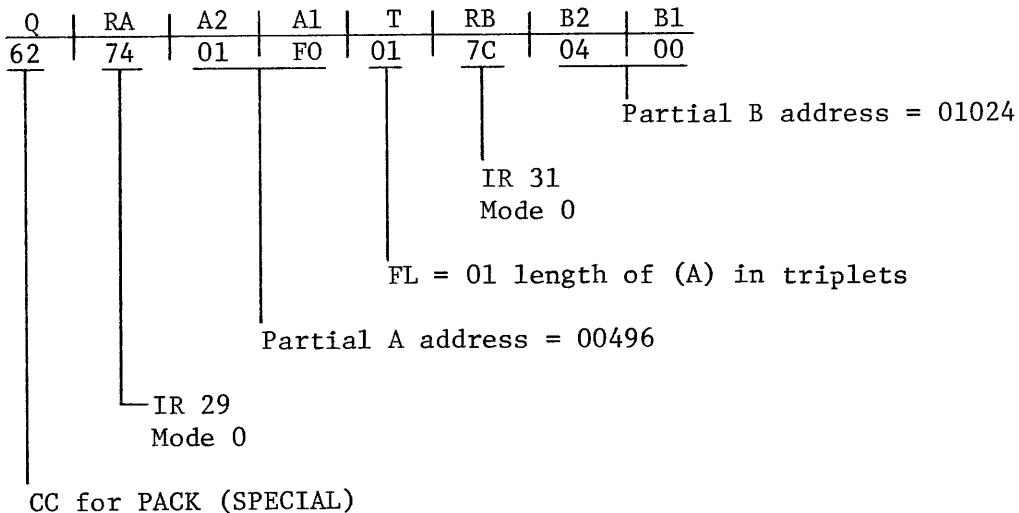
Q	RA	A2	A1	T	RB	B2	B1
62	74	01	F0	01	7C	04	00

Partial B address = 01024

IR 31
Mode 0

FL = 01 length of (A) in triplets

Partial A address = 00496

IR 29
Mode 0

CC for PACK (SPECIAL)

Assume the index registers contain the following:

    IR29 = 0B54   (02900 decimal)
    IR31 = 0C1C   (03100 decimal)

After command setup:

Effective A address = 0B54 + 01F0 = 0D44 (03396 decimal)
Effective B address = 0C1C + 0400 = 101C (04124 decimal)

Before command execution:

A	0D44	0D45	0D46
(A)	0101 1111	0011 0101	0010 0100

The contents of B are not significant.

(A) | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

(B) | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |

315 slab | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
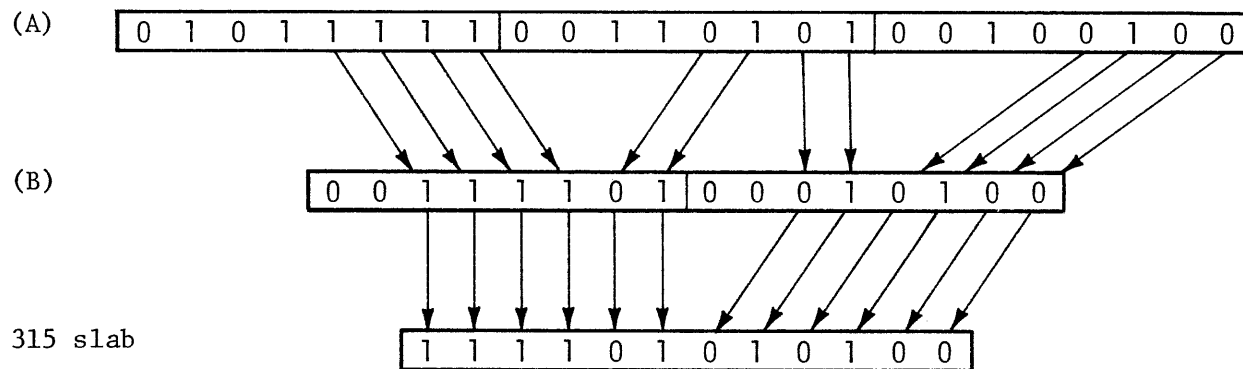
The PACK command is used to pack information to be used in a 315 mode. There-
fore, the example illustrates (A) being packed and transferred to (B). If (B)
is to be used by the 315 simulator option it is considered to be a 315 slab as
illustrated.

$$C = 63(3F)(BF)$$

The six low-order bits of each pair of (A) are considered to be three 4-bit digits and are stored into three characters of (B), starting with the leftmost character of each field.

The four high-order bits of (B) are set to 0.

The beginning address of the A field must be evenly divisible by 2 or a PE occurs. T specifies the length of (A) in pairs of characters and ranges from 0 through 255, with 0 = 256.

$\underline{B} = B + 3T$, except when T = 0, then $\underline{B} = B + 768$
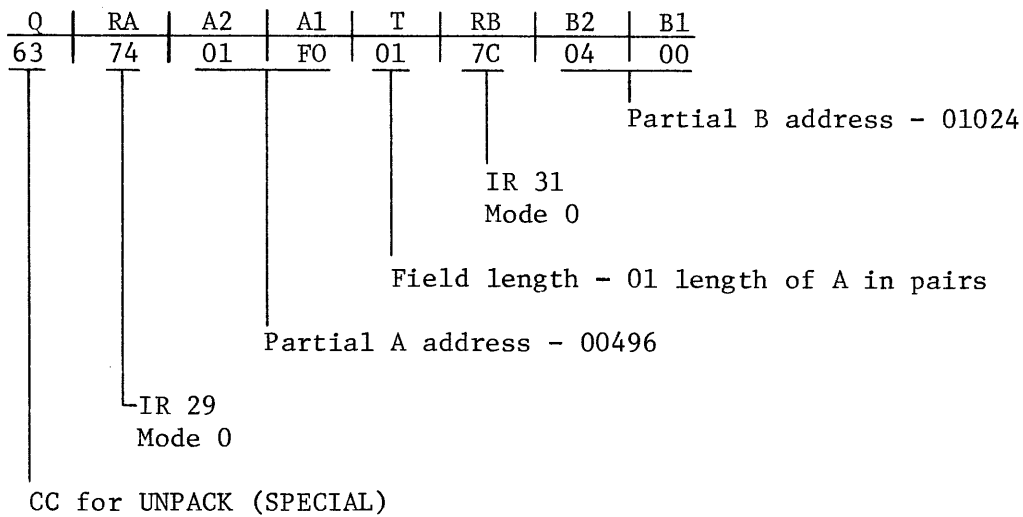
Q	RA	A2	A1	T	RB	B2	B1
63	74	01	FO	01	7C	04	00

Partial B address - 01024

IR 31
Mode 0

Field length - 01 length of A in pairs

Partial A address - 00496

IR 29
Mode 0

CC for UNPACK (SPECIAL)

Assume the index registers contain the following:

    (IR29) = 0B54   (02900 decimal)
    (IR31) = 0C1C   (03100 decimal)

Effective A address = 0B54 + 01F0 = 0D44 (03396 decimal)
Effective B address = 0C1C + 0400 = 101C (04124 decimal)

Before command execution:

A	0D44	0D45
(A)	0011 1111	0011 1111

The contents of B are not significant.

(A) `0 0 1 1 1 1 1 1 0 0 1 1 1 1 1 1`

(B) `0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 1 1 1 1`
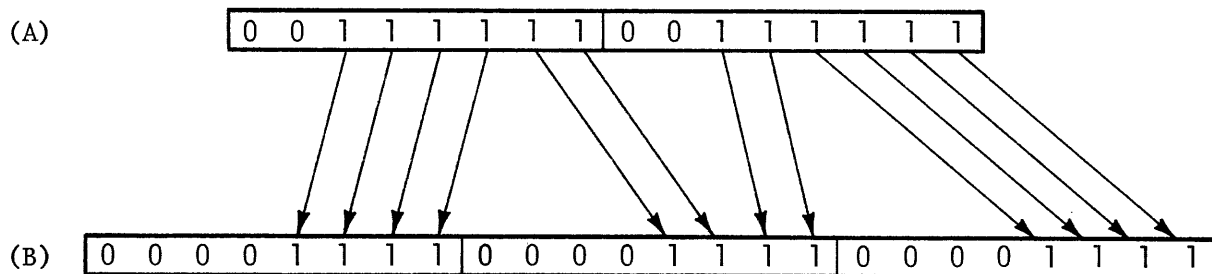
The multiprogramming option includes one additional machine command, LOAD BAR. For a detailed function description of this option refer to the NCR Century 200 Processor publication (ST-9402-13).

LOAD BAR

$C = 88(58)(D8)$

The effective A address specifies a 4-character field, of which only the two rightmost characters are used. Of those two, the leftmost character specifies the value to be loaded into the BAR, and the rightmost character specifies the value to be loaded into LAR. Only the five low-order bits of each character are used (eight bits in systems having the extended memory option).

If the S Flag is not ON or if the effective A address is not evenly divisible by 4 a PE results and the BAR and LAR remain undisturbed.

T and B are not used.

NOTE

The LOAD BAR command is only used with the multiprogramming option and should not be confused with a register (BAR) of the 315 simulator option.

The trace option includes three additional machine commands: LOAD MONITOR
REGISTER, LOAD TRACE, and STORE TRACE. For a detailed functional description
of this option refer to the NCR Century 200 Processor publication (ST-9402-13).

LOAD MONITOR REGISTER

$$C = 88(59)(D9)$$

The effective A address specifies a 4-character field, of which only the 19
rightmost bits are used. These 19 bits constitute an address which is entered
into the monitor register.

If the effective A address is not evenly divisible by 4, a PE results and the
monitor register is not disturbed.

B and T are not used.

LOAD TRACE

$$C = 90(5A)(DA)$$

The character specified by the effective A address is tested. If the b4 posi-
tion of the character is a 1 bit, the trace permit is set ON. If the b4 posi-
tion of the character is a 0 bit, the trace permit is set OFF. In either case,
the remaining 7 bits of the character are ignored.

B and T are not used.

STORE TRACE

$$C = 91(5B)(DB)$$

The trace permit is tested. If it is ON, it is set OFF and a 1 bit is stored
in the b4 position of the character specified by the effective A address. If
it is OFF, a 0 bit is stored in the b4 of the character specified by the effec-
tive A address. In either case the remaining 7 bits of the character are set
to zeros.

B and T are not used.

NOTE

This command cannot be traced and the address into which
the b4 indicator is stored cannot be monitored by the
monitor address register.

C = 92(5C)(DC)

TABLE COMPARE is a combination of the DECODE ALL and COMPARE BINARY commands.

Each character of the A-field and the corresponding character of the B-field are decoded according to a table and the resulting characters are then compared. The original characters of A and B are left undisturbed in memory. The operation occurs sequentially from the leftmost character, terminating at the first pair of decoded characters which are unequal or when the fields are exhausted.

The decoding is as follows:

For a given character, (A1 or B1), the associated table character is located in address 282-283 + K1, where K1 is considered to be a 2-character binary address in which the left character is zero and the right character is either A1 or B1.

The decoded A character is binarily compared to the decoded B character. If the decoded characters are equal the command repeats the decoding and comparing with the next A and B characters (E flag ON). If the characters are unequal, whether the B field is exhausted or not, the command terminates with the L or G flag ON. If the characters are equal and the B field is exhausted, the command terminates with the E flag ON.

If the E or G flag is turned ON, the secondary repeat indicator (RII) is turned OFF.

T specifies the length of both A and B and ranges from 0 to 255, with 0 equivalent to 256.

With the E flag ON:
$\underline{B}$ = B + T except when T = 0, then $\underline{B}$ = B + 256.

With the L or G flag ON:
$\underline{B}$ = the address + 1 of the character in B causing termination.

Command Execution Time

    E = PO + 5N(P)
    N = Number of pairs of characters decoded to find equal compare.
        $1 \leq N \leq T$.

```
    Q  |  RA  |  A2  |  A1  |  T  |  RB  |  B2  |  B1
   5C  |  64  |  01  |  F4  |  05 |  40  |  00  |  FA
    |      |      |     |      |     |      |
    |      |      |     |      |     |      |        Partial B address = 00250
    |      |      |     |      |     |
    |      |      |     |      |     |      IR 16
    |      |      |     |      |     |      Mode 0
    |      |      |     |      |
    |      |      |     |      |    FL = 5
    |      |      |     |
    |      |      |    Partial A address = 00500
    |      |
    |      └─IR 25
    |         Mode 0
    |
   CC for 2-address TABLE COMPARE command
```
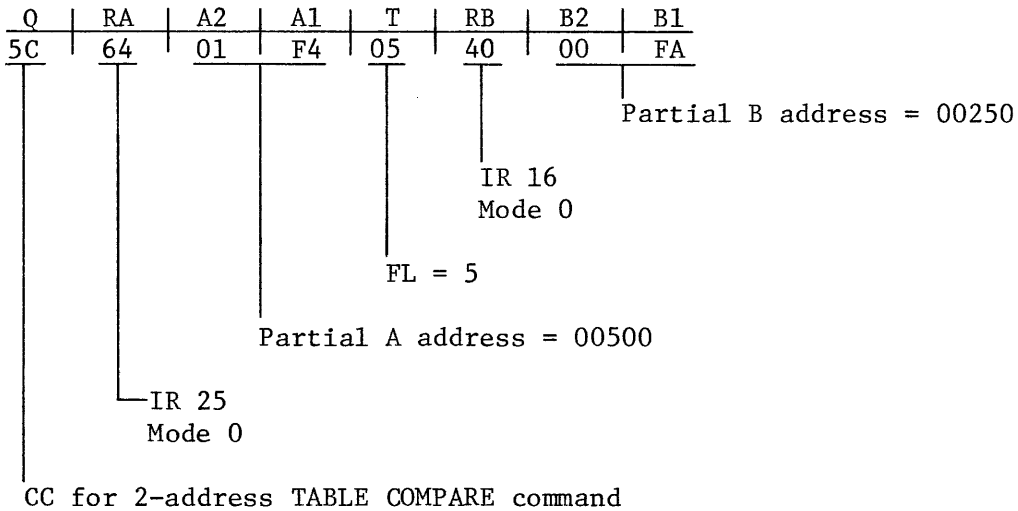
Assume the index registers contain the following:

    (IR16) = 07D0 (02000)
    (IR25) = 0BB8 (03000)

After command setup:

Effective A address = 07D0 + 01F4 = 09C4 (02500)
Effective B address = 0BB8 + 00FA = 0CB2 (03250)

The A and B operand contents remain unchanged.

A	09C4	09C5	09C6	09C7	09C8
(A)	0000 0000	0000 0101	0000 0100	0000 0010	0000 0001

B	0CB2	0CB3	0CB4	0CB5	0CB6
(B)	0000 0100	0000 0001	0000 0010	0000 0011	0000 0101

011A (00282)

011A	011B
14	A0

14A0 (05280)   DECODE TABLE

14A0	14A1	14A2	14A3	14A4	14A5
0100 0100	0100 0101	0100 0011	0100 1111	0100 0100	0100 0101
D	E	C	0	D	E

ASCII CODE

1. 14A0   The leftmost character of the A operand is added to the
    + 00   contents of memory location 00283. The resulting memory
    14A0   address is read out.

2. 14A0   The leftmost character of the B operand is added to the
    + 04   contents of memory location 00283. The resulting memory
    14A4   address is read out.

3. | 14A0 | = | 14A4 | The contents of the two addresses
    | 0100 0100 | | 0100 0100 | are compared.

The process is repeated, using succeeding A and B operand characters,
until the characters are unequal or the field is exhausted.

1. 14A0   2. 14A0   3. | 14A5 | = | 14A1 |
    + 05   + 01 | 0100 0101 | | 0100 0101 |
    14A5   14A1

1. 14A0   2. 14A0   3. | 14A4 | > | 14A2 |
    + 04   + 02 | 0100 0100 | | 0100 0011 |
    14A4   14A2

The command terminates and the G flag is ON.

$$C = 93(5D)(DD)$$

The contents of the field specified by the effective B address (multiplicand) are decimally multiplied by the contents of the field specified by the effective A address (multiplier). Each field is considered to contain signed, packed (see PACK command description) decimal information; the multiplication is algebraic.

If (A) and (B) have like signs, a plus sign configuration (1011) is stored for a positive result. If (A) and (B) have unlike signs, a minus sign configuration (1101) is stored for a negative result.

The result (product) is placed right justified in a fixed area of memory called the memory accumulator. The accumulator is 16 characters (32 digits) in length, occupying locations 320-335 (hex 140-14F). The characters in the accumulator to the left of the product remain undisturbed. The length of the product is equal to the sum of the lengths of the (A) and (B) operands.

T is treated as two 3-bit bytes, with $T_7 - T_5$ specifying the length of (A) and $T_3 - T_1$ specifying the length of (B). These lengths are expressed as the number of characters per field and range in value from 0 through 7, with 0 equivalent to 8. T_8 and T_4 must be equal to zero or a PE will occur.

If (A) or (B) overlaps any part of the product area, the result may be different from the result obtained when (A) or (B) does not overlap the product area.

If any digit of (A) or (B) other than the sign digit contains a value other than 0-9, a PE occurs with indeterminate results in the accumulator.

NOTE

Following command execution the product is stored at the address $335 - (T_A + T_B)$. Since T after command execution is not equal to the length of the product, the MULTIPLY command must not be followed by a 1-address command.

Command Execution Time

$$E = 2 \text{ PO } (TA + 1) + 3P + 17 \text{ TBP} + X [P_0 + 4(TBP + 1P) + 1P] +$$

$$Y [P_0 + 3 (TBP + 1P) + 1P]$$

TA = Length of the A Field as expressed by bits T_{5-8} of the command.

TB = Length of the B Field as expressed by bits T_{1-3} of the command.

X = Value derived from summing the weights associated with the most significant digit of each character in the A Field.

Y = Value derived from summing the weights associated with the least significant digit of each character from the A Field.

To illustrate, the diagram below represents an A field of length TA = 4

$$X_4 \ Y_3 \qquad X_3 \ Y_2 \qquad X_2 \ Y_1 \qquad X_1 \ \text{SIGN}$$

The term X is computed by summing the weights assigned the terms X_1, X_2, X_3, and X_4 according to the value of the digits they represent as determined by the table below.

X TABLE	
DIGIT VALUE	WEIGHT
0	0
1,2,4,8	1
3,5,6,7,9	2

If $X_1 = 4$, $X_2 = 0$, $X_3 = 7$, and $X_4 = 3$ then X = 1 + 0 + 2 + 2 = 5

The value of Y is derived from the weight assigned the terms Y_1, Y_2, and Y_3 according to the value of the digits they represent in the table below.

Y TABLE	
DIGIT VALUE	WEIGHT
0,1,2,4,8	1
3,5,6,7,9	2

EXAMPLE 1 (Like Signs)
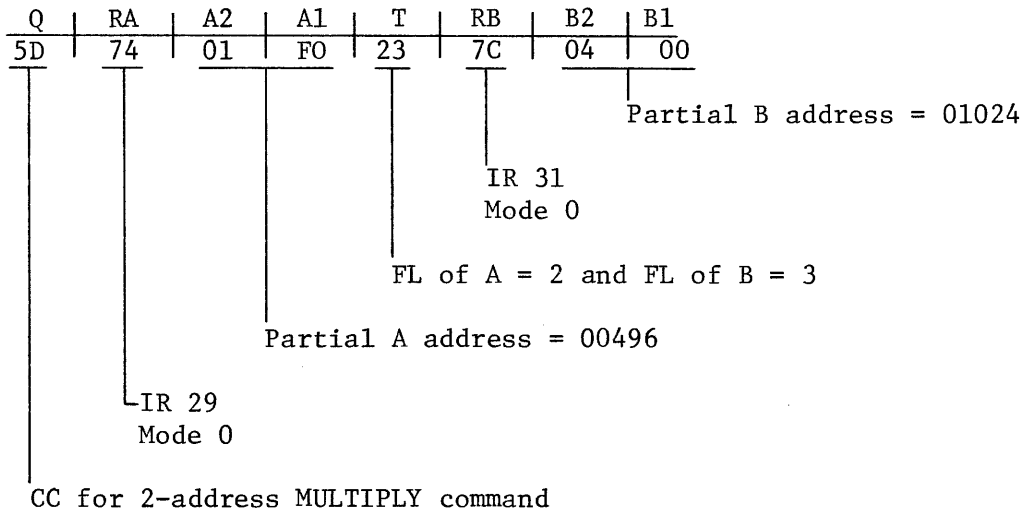
```
   Q  | RA | A2 | A1 |  T | RB | B2 | B1
   5D | 74 | 01 | FO | 23 | 7C | 04 | 00
```

Partial B address = 01024

IR 31
Mode 0

FL of A = 2 and FL of B = 3

Partial A address = 00496

IR 29
Mode 0

CC for 2-address MULTIPLY command

Assume the index registers contain the following:

    (IR29) 0B54 (02900)
    (IR31) 0C1C (03100)

After command setup:

Effective A address = 0B54 + 01F0 = 0D44 (03396)
Effective B address = 0C1C + 0400 = 101C (04124)

Before command execution:

```
  A  |   0D44    |    0D45    |
 (A) | 0000 0001 | 0101 1011 |
                        sign
```
Packed BCD value = 15 +

```
  B  |   101C    |    101D    |   101E    |
 (B) | 0000 0001 | 1000 0101 | 0000 1011 |
                                   sign
```
Packed BCD value = 1850 +

Decimal equivalent of multiplication to be performed:

$$
\begin{array}{r}
1850\ + \\
\underline{\times\ 15}\ + \\
9250 \\
\underline{1850} \\
27750
\end{array}
$$

After command execution:

Accumulator

014B	014C	014D	014E	014F
0000 0000	0000 0000	0010 0111	0111 0101	0000 1011

(Accumulator)

sign

Packed BCD value = 27750 +

<u>NOTE</u>

The contents of accumulator locations lower than 014B are unchanged by this operation.

EXAMPLE 2 (Unlike Signs)

Assume that the fields used in the preceding example had contained the following information:

Before command execution:

A (A)	0D44	0D45				Packed BCD value = 15 −
	0000 0001	0101 1101				

sign

B (B)	101C	101D	101E			Packed BCD value = 1850 +
	0000 0001	1000 0101	0000 1011			

sign

Decimal equivalent of multiplication to be performed:

```
    1850 +
  x  15 -
    9250
    1850
   27750 -
```

After command execution:

Accumulator (Accumulator)	014B	014C	014D	014E	014F
	0000 0000	0000 0000	0010 0111	0111 0101	0000 1101

sign

Packed BCD value = 27750 −

<u>NOTE</u>

The contents of accumulator locations lower than 014B are unchanged by this operation.

C = 94 (5E) (DE)

The LOGIC command uses the 1-byte A and B operands to perform the logic function specified by the T character of the command. The result of the operation replaces the B operand.

The logic operations, based on Boolean algebra, are performed on the individual, corresponding bit positions of the A and B operands. For example, if the T character specifies a logical sum (B = A + B), then B (bit 1) = A (bit 1) + B (bit 1), B (bit 2) = A (bit 2) + B (bit 2), etc.

The four least significant bits (b1-b4) of the T character specify one of 16 logic functions to be performed. (The four most significant bits of the T character are not used, but they should be zero to be compatible with other processors of the NCR Century Series that use them.)

FUNCTIONS OF LOGIC COMMAND	
T $(b_4 - b_1)$ HEX VALUE	FUNCTION
0	B = 0 (clear B)
1	B = (A+B)' or A'B'
2	B = A'B
3	B = A'
4	B = AB'
5	B = B'
6	B = AB' + A'B
7	B = (AB)' or A'+B'
8	B = AB
9	B = A'B' + AB
A	B = B (store B)
B	B = A' + B
C	B = A (store A)
D	B = A + B'
E	B = A + B
F	B = 1 (set B)

Q	R_A	A_2	A_1	T	R_B	B_2	B_1
5E	00	04	40	08	00	08	90

Effective B address

No indexing

Function: "LOGICAL PRODUCT" -- 08

Effective A address

No indexing

CC for 2-address LOGIC command

Before command execution:

A
| 0440 | 1 1 0 1 0 0 1 1 | (A)

B
| 0890 | 0 1 1 1 1 0 0 1 | (B)

B = AB

AB = 1, replaces B (bit 1)

AB = 0, replaces B (bit 2)

AB = 0, replaces B (bit 3)

AB = 0, replaces B (bit 4)

AB = 1, replaces B (bit 5)

AB = 0, replaces B (bit 6)

AB = 1, replaces B (bit 7)

AB = 0, replaces B (bit 8)

After command execution:

B
| 0890 | 0 1 0 1 0 0 0 1 | (B)

Q	R_A	A_2	A_1	T	R_B	B_2	B_1
5E	00	04	4C	06	00	08	98

Effective B address

No indexing

Function: "EXCLUSIVE OR" -- 06

Effective A address

No indexing

CC for 2-address LOGIC command

Before command execution:

A
| 044C | | 1 0 1 0 1 0 1 0 | (A)

B
| 0898 | | 0 0 0 0 1 1 1 1 | (B)

$B = AB' + A'B$

AB' + A'B = 1, replaces B (bit 1)

AB' + A'B = 0, replaces B (bit 2)

AB' + A'B = 1, replaces B (bit 3)

AB' + A'B = 0, replaces B (bit 4)

AB' + A'B = 0, replaces B (bit 5)

AB' + A'B = 1, replaces B (bit 6)

AB' + A'B = 0, replaces B (bit 7)

AB' + A'b = 1, replaces B (bit 8)

After command execution:

B
| 0898 | | 1 0 1 0 0 1 0 1 | (B)

INTRODUCTION

The floating point option for the NCR Century 200 System automatically scales the numbers involved in a computation and maintains the precision of the result of the computation. The option comprises 12 commands that provide for addition, subtraction, comparison, multiplication, multiplication-addition, and division.

GENERAL

A floating point number is expressed as the product of a signed fraction and a base number raised to a power:

$$.A \times B^n$$

Where:

.A is the signed fraction, having a radix point to the left of its leftmost digit.

B is the base determined by the particular number system (10 for decimal, 2 for binary, 16 for hexadecimal, etc.)

n is the characteristic, or power to which the base is raised.

FLOATING POINT FORMAT

Floating point numbers occupy fixed-length formats as either 4- or 8-byte fields:

Single Precision

4 BYTES

```
┌────┬───────┬──────┬──────┬──────┐
│ b8 │ b7─b1 │   |  │   |  │   |  │
└────┴───────┴──────┴──────┴──────┘
                └─── FRACTION (6 HEXADECIMAL DIGITS)
        └CHARACTERISTIC
   └─SIGN OF THE FRACTION
```

Double Precision

8 BYTES
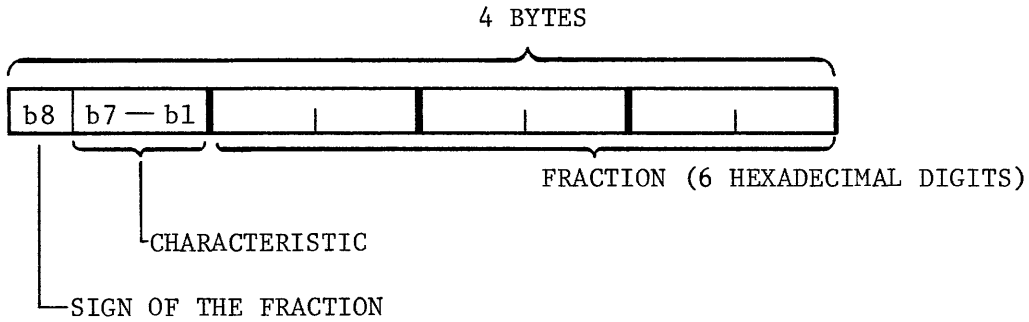
```
┌────┬───────┬──────┬──────┬─┬──────┐
│ b8 │ b7─b1 │   |  │   |  ≀ │   |  │
└────┴───────┴──────┴──────┴─┴──────┘
                └─── FRACTION (14 HEXADECIMAL DIGITS)
        └CHARACTERISTIC
   └─SIGN OF THE FRACTION
```
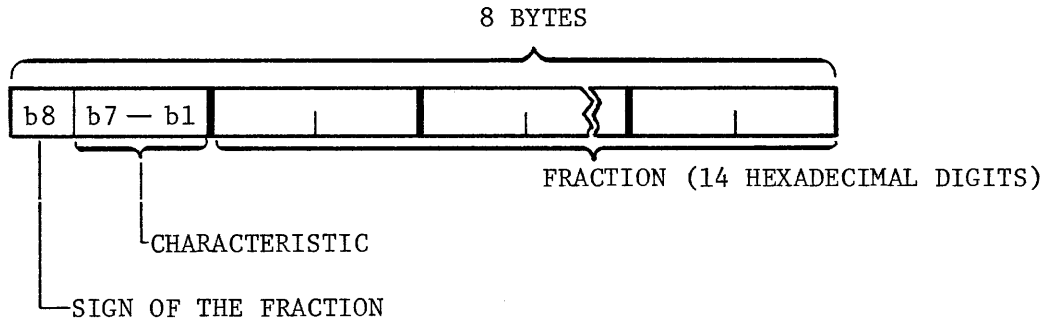
Sign of the Fraction

If b8 = 0, the sign is positive.
If b8 = 1, the sign is negative.

Characteristic

The characteristic is a 7-bit hexadecimal number. The characteristic specifies the exponent value and its sign. (The exponent is the power to which the base is raised).

	DECIMAL	BINARY	HEXADECIMAL
$.A \times B^{+63}$	+ 63	111 1111	7F
$.A \times B^{0}$	0	100 0000	40
$.A \times B^{-64}$	- 64	000 0000	00

A hexadecimal 40 in the characteristic position expresses an exponent value of 0. Any value above hexadecimal 40 indicates a positive exponent and any value below indicates a negative exponent.

The characteristic may range in value from -64 to +63.

The amount of the characteristic above hexadecimal 40 determines where the radix point is located to the right of the leftmost digit when the floating point fraction is expressed in hexadecimal notation.

FLOATING POINT   HEXADECIMAL NOTATION

| 4 5 | 1 2 | 3 4 | 5 6 | $= .123456 \times 16^{5} = 12345.6$
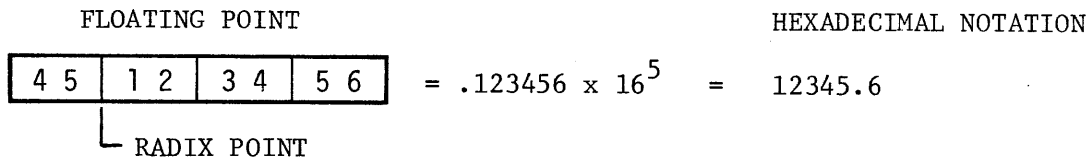
    └─ RADIX POINT

The amount of the characteristic below hexadecimal 40 determines where the radix point is located to the left of the leftmost digit when the floating point fraction is expressed in hexadecimal notation.

FLOATING POINT   HEXADECIMAL NOTATION

| 3 7 | 1 2 | 3 4 | 5 6 | $= .123456 \times 16^{-3} = .000123456$
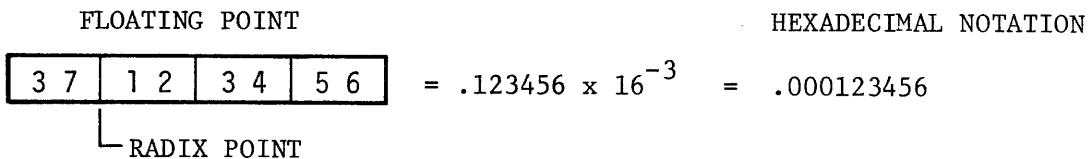
    └─ RADIX POINT

Fraction

The fraction consists of six (single precision) or 14 (double precision) hexadecimal digits. The radix point is immediately to the left of the leftmost digit of the fraction, and the characteristic is adjusted accordingly. The number may be normalized; that is, have no leading zeros between the most significant digit and the radix point.

The decimal number 31046.5 expressed in hexadecimal notation is 7946.8. It is shown in floating point format below.

NORMALIZED

4 4	7 9	4 6	8 0

└─RADIX POINT

$= .79468 \times 16^4$

NON-NORMALIZED

4 5	0 7	9 4	6 8

└─RADIX POINT

$= .079468 \times 16^5$

FLOATING POINT FRACTIONAL NUMBER REPRESENTATION AND CONVERSION

Representation

Fraction floating point numbers are hexadecimal expressions which when taken
out of floating point format (characteristic adjusted to a hexadecimal 40, or
CO if a negative representation, and the radix point moved accordingly), have
one or more digits greater than zero to the right of the radix point.

Two types of fractional numbers may be represented in floating point notation,
mixed numbers and pure fractions.

● Mixed Numbers

   Mixed number representations result in hexadecimal expressions that have
   one or more digits to the left and right of the radix point when taken out
   of floating point format. This type of hexadecimal expression represents
   a decimal number that consists of a whole number and decimal fraction as
   in the following single precision floating point representation.

Non-normalized --------------------------	45.01A2F4
Normalized ----------------------------	44.1A2F40
Which is the same as --------------------	$.1A2F40 \times 16^4$
When removed from floating point format =	$1A2F.40 \times 16^0$
	$1A2F.40 \times 1$
Which in binary is expressed as -------	0001101000101111.01000000
Having a decimal value of --------------	6703.250

<div align="center">NOTE</div>

   The actual method of conversion from hexadecimal to a decimal
   value is shown on following pages.

● Pure Fraction Numbers

   Pure fraction representations result in hexadecimal expressions that have one
   or more digits to the right of the radix point when taken out of floating
   point format. This type of hexadecimal expression represents a decimal frac-
   tion as in the following single precision floating point representation.

Non-normalized ----------------------	41.02A310
Normalized -------------------------	40.2A3100
Which is the same as ----------------	$.2A3100 \times 16^0$
When removed from floating point format --	$.2A3100 \times 1$
Which in binary is expressed as ----	0.00101010001100010000000
Having a decimal value of ----------	0.1700439453125

Conversion

Floating point representations of positive or negative decimal numbers (whole numbers, mixed numbers, fractional numbers) may be converted to their decimal equivalent by taking the expression out of floating point format, transposing it to its binary form, maintaining the radix point, and summing up the decimal values of the 1 bits according to their binary position.

1. Given the double precision floating point number

   46.4ADEAF31BABE00

2. Take the number out of floating point format (adjust the radix point to a characteristic of 40, or C0 if negative).

   4ADEAF.31BABE00

3. Convert each hexadecimal digit to its binary configuration, maintaining the radix point at its proper position.

0100	1010	1101	1110	1010	1111	.	0011	0001	1011	1010	1011	1110	0000	0000
4	A	D	E	A	F	.	3	1	B	A	B	E	0	0
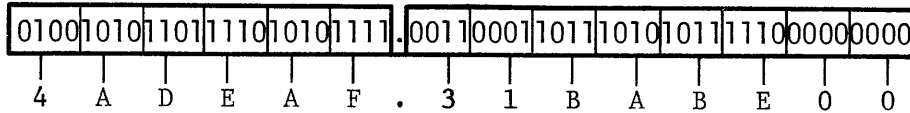
4. Add the decimal values of the 1 bits according to the binary (powers of 2) table given on page 131, maintaining each value in its proper relation to the radix point.

   The power of 2 is increased positively by 1 for each bit position to the left of the radix point. Bit position 1 = a decimal value of 2^0 = 1. Bit 2 = a decimal value of 2^1 = 2, etc. For bit positions to the right of the radix point, the power of 2 is increased negatively by 1 for each bit position. The first bit position to the right of the radix point = a decimal value of 2^{-1} = .500, bit 2 = a decimal value of 2^{-2} = .250 etc.

Starting with the decimal values for 1 bits to the left of the radix point, the addition for the example given is:

```
           1.
           2.                    '
           4.
           8.
          32.
         128.
         512.
        1024.
        2048.
        4096.
       16384.
       32768.
      131072.
      524288.
     4194304.                          1 bit values to the left of radix point
            .125                       1 bit values to the right of radix point
            .0625
            .00390625
            .001953125
            .00048828125
            .000244140625
            .0001220703125
            .000030517578125
            .00000762939453125
            .0000019073486328125
            .00000095367431640625
            .000000476837158203125
            .0000002384185791015625
            .00000011920928955078125
    5906671.19425570964813232421875
```

5. Adopt the sign designated by the sign bit of the floating point representation (0 bit = +, 1 bit = -).

Therefore, the floating point representation: 46.4ADEAF31BABE00 has a decimal equivalent of + 5906671.19425570964813232421875.

POWERS OF 2 TABLE (PARTIAL)

2^n	n	2^{-n}
1	0	1.0
2	1	0.5
4	2	0.25
8	3	0.125
16	4	0.062 5
32	5	0.031 25
64	6	0.015 625
128	7	0.007 812 5
256	8	0.003 906 25
512	9	0.001 953 125
1 024	10	0.000 976 562 5
2 048	11	0.000 488 281 25
4 096	12	0.000 244 140 625
8 192	13	0.000 122 070 312 5
16 384	14	0.000 061 035 156 25
32 768	15	0.000 030 517 578 125
65 536	16	0.000 015 258 789 062 5
131 072	17	0.000 007 629 394 531 25
262 144	18	0.000 003 814 697 265 625
524 288	19	0.000 001 907 348 632 812 5
1 048 576	20	0.000 000 953 674 316 406 25
2 097 152	21	0.000 000 476 837 158 203 125
4 194 304	22	0.000 000 238 418 579 101 562 5
8 388 608	23	0.000 000 119 209 289 550 781 25
16 777 216	24	0.000 000 059 604 644 775 390 625
33 554 432	25	0.000 000 029 802 322 387 695 312 5
67 108 864	26	0.000 000 014 901 161 193 847 656 25
134 217 728	27	0.000 000 007 450 580 596 923 828 125
268 435 456	28	0.000 000 003 725 290 298 461 914 062 5
536 870 912	29	0.000 000 001 862 645 149 230 957 031 25
1 073 741 824	30	0.000 000 000 931 322 574 615 478 515 625
2 147 483 648	31	0.000 000 000 465 661 287 307 739 257 812 5
4 294 967 296	32	0.000 000 000 232 830 643 653 869 628 906 25
8 589 934 592	33	0.000 000 000 116 415 321 826 934 814 453 125
17 179 869 184	34	0.000 000 000 058 207 660 913 467 407 226 562 5
34 359 738 368	35	0.000 000 000 029 103 830 456 733 703 613 281 25
68 719 476 736	36	0.000 000 000 014 551 915 228 366 851 806 640 625
137 438 953 472	37	0.000 000 000 007 275 957 614 183 425 903 320 312 5
274 877 906 944	38	0.000 000 000 003 637 978 807 091 712 951 660 156 25
549 755 813 888	39	0.000 000 000 001 818 989 403 545 856 475 830 078 125
1 099 511 627 776	40	0.000 000 000 000 909 494 701 772 928 237 915 039 062 5

CONVENTIONS

All commands except FLOATING POINT COMPARE SINGLE and FLOATING POINT COMPARE DOUBLE produce normalized results; therefore, any floating point number can be normalized by adding it to floating point zero.

Since only hexadecimal digits can be normalized, the three leftmost bits of a normalized fraction may be zero.

Although single precision floating point words have a 6-digit fraction, intermediate results in addition, subtraction, and division may be expressed with 7-digit fractions. The seventh (rightmost) digit is a guard digit that increases the precision of the final result. This guard digit is not stored in memory at the termination of the operation, nor is one used in double precision operations.

Add and subtract commands replace the original B operand with the results of the computation. No other commands disturb the A and B operands in memory, and, except for those commands that store into a memory accumulator, no memory character other than the referenced floating point words is ever affected.

A positive number with a zero characteristic and a zero fraction is a true zero. A true zero may result because of the particular magnitude of the operands. A result is forced to be true zero when an exponent underflow occurs or when a result fraction is zero.

If a result has a zero fraction, overflow of the characteristic will not cause the PE trap that would otherwise occur.

If a divisor has a zero fraction, division is inhibited and a PE trap occurs. Otherwise, zero fractions and zero characteristics are treated as normal numbers in all arithmetic operations.

A zero fraction will not result from any operation except as part of a true zero; if a number having a zero fraction and a non-zero characteristic is introduced, an incorrect operation may result.

FLOATING POINT PROGRAMMER ERROR CODES	
ERROR	CODE (STORED IN LOCATION 36)
F.P. Number Address $\neq$ 0 Modulo 4	1
Characteristic Overflow [Characteristic $> +$ 63]	5
Attempting To Divide By 0	1

C = 116 (74)(F4)

The contents of the field specified by the effective A address are added to the contents of the field specified by the effective B address. Each field is considered to contain a floating point, single precision number (2-digit sign/characteristic and 6-digit fraction); addition is performed on the fractions. The result of the addition is an intermediate sum which is normalized and truncated to give a final sum. The final sum replaces (B).

In performing the operation, the characteristics of the two operands are compared. If the difference is less than seven, the fraction with the smaller characteristic is right-shifted the number of hexadecimal digits specified by that difference. The fractions are then added algebraically to form the intermediate sum. This includes checking signs and complementary addition (subtraction) if the signs are unlike. The larger of the two characteristics is subsequently referenced. If a fraction overflow carry results from the addition, the intermediate sum is right-shifted one digit to make room for the carry and the characteristic is increased by one. If this increase causes a characteristic overflow, a PE trap occurs with the erroneous result stored in (B).

If the difference between the characteristics of the two operands is seven or greater, the fraction with the larger characteristic automatically becomes the intermediate sum. This is due to the fact that, had the fraction with the smaller characteristic been shifted to the right seven positions, zeros would have been added to the fraction with the larger characteristic, giving the same result. The larger characteristic is subsequently referenced.

Before addition, a fraction which has been shifted right can be considered to contain seven hexadecimal digits. The rightmost (seventh) digit is a guard digit and participates in the addition. If no shift occurs, a guard digit of zero is assumed. The resulting intermediate sum of the fractions also consists of seven hexadecimal digits, including the possible overflow carry. If these digits are all zero, a true zero (32 zero bits) is generated for the final result (sign, characteristic, and fraction all zeros).

If a true zero is not generated, the intermediate sum of the fractions, including the guard digit, is left-shifted as necessary to form a normalized fraction. Vacated low order (rightmost) digit positions are filled with zeros and the characteristic is reduced by the amount of the shift. If this normalization causes the characteristic to underflow, a true zero is generated (32 zero bits), and the overflow indicator (OF) is set ON.

Finally, the intermediate sum is truncated to the proper length (rightmost digit removed) and the result with its characteristic (32 bits altogether) is stored in (B).

The sign of the sum is derived by the rules of algebra. A zero fraction cannot result, except when a true zero results or is generated, in which case the sign is always positive.

One of the L, E, or G flags is set ON reflecting the results: less than, equal to, or greater than relationship to zero.

T is not used unless this command is to be followed by a 1-address command. In this case, the T specified would be retained for use in the following command but would have no effect on this command. The length of B after execution is equal to the length of B before execution.

Command Execution Time

$$E = 2\,PO + 12P + GP + K\,(2\,PO + 8P) + X\,(2\,PO + 9P + \frac{1 + N}{2}P) +$$

$$Y\,(2\,PO + 10P + \frac{M}{2}P)$$

G = 1   If the exponent difference is non-zero but within the range where the fractions must be added, otherwise G = 0.

K = 1   If the result fraction overflows, otherwise K = 0.

X = 1   If normalization is required on the intermediate result and the field contains an odd number of leading zeros.

N =   Number of odd leading zeros.

Y = 1   If normalization is required on the intermediate result and the field contains an even number of leading zeros.

M =   Number of even leading zeros.

X and Y are mutually exclusive.

HEXADECIMAL ADDITION TABLE

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	c0
2	2	3	4	5	6	7	8	9	A	B	C	D	E	F	c0	c1
3	3	4	5	6	7	8	9	A	B	C	D	E	F	c0	c1	c2
4	4	5	6	7	8	9	A	B	C	D	E	F	c0	c1	c2	c3
5	5	6	7	8	9	A	B	C	D	E	F	c0	c1	c2	c3	c4
6	6	7	8	9	A	B	C	D	E	F	c0	c1	c2	c3	c4	c5
7	7	8	9	A	B	C	D	E	F	c0	c1	c2	c3	c4	c5	c6
8	8	9	A	B	C	D	E	F	c0	c1	c2	c3	c4	c5	c6	c7
9	9	A	B	C	D	E	F	c0	c1	c2	c3	c4	c5	c6	c7	c8
A	A	B	C	D	E	F	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
B	B	C	D	E	F	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	cA
C	C	D	E	F	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	cA	cB
D	D	E	F	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	cA	cB	cC
E	E	F	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	cA	cB	cC	cD
F	F	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9	cA	cB	cC	cD	cE

c = Carry

EXAMPLE 1 (Like Signs)

Q	RA	A2	A1	T	RB	B2	B1
74	74	01	FO	00	7C	04	00

Partial B address = 01024

IR 31
Mode 0

FL = NOT USED

Partial A address = 00496

IR 29
Mode 0

CC for 2-address FLOATING POINT ADD SINGLE command

Assume the index registers contain the following:

   (IR 29) = OB54 (02900)
   (IR 31) = OC1C (03100)

After command setup:

Effective A address = OB54 + 01FO = OD44 (03396)
Effective B address = OC1C + 0400 = 101C (04124)

Before command execution:

A	OD44	OD45	OD46	OD47
(A)	0100 0010	0100 0001	0010 0100	0011 0110

Hexadecimal value = 42412436

B	101C	101D	101E	101F
(B)	0100 0000	0011 0110	0010 0010	0011 0100

Hexadecimal value = 40362234

	SIGN 0=+ 1=−	CHARACTERISTIC	FRACTION	GUARD DIGIT
(A)	0	100 0010 — 4 2	0100 0001 0010 0100 0011 0110 — 4 1 2 4 3 6	
(B)	0	100 0000 — 4 0	0011 0110 0010 0010 0011 0100 — 3 6 2 2 3 4	

Step 1:
Calculate the difference between the characteristics.

		000 0010 — 0 2		

Step 2:
The difference is less than 7. Shift the fraction with the smaller characteristic right the number of positions specified by the difference.

(B)			0011 0110 0010 0010 0011 0100 — 3 6 2 2 3 4	
(B)			0000 0000 0011 0110 0010 0010 — 0 0 3 6 2 2	0011 — 3

Step 3:
Compare signs. If unlike, next step is to subtract fractions. If like, next step is to add fractions.

(A)	0			
(B)	0			
	0		Like signs	

Step 4:
Add fractions to form intermediate sum. The larger of the two characteristics is subsequently referenced.

(A)			0100 0001 0010 0100 0011 0110 — 4 1 2 4 3 6	
(B)			0000 0000 0011 0110 0010 0010 — 0 0 3 6 2 2	0011 — 3
		100 0010 — 4 2	0100 0001 0101 1010 0101 1000 — 4 1 5 A 5 8	0011 — 3

Step 5:
Normalize the intermediate sum and adjust the characteristic.

Already normalized

Step 6:
Truncate the result (drop rightmost digit) and store in (B). The sign is governed by the rules of algebra.

(B)	0	100 0010 — 4 2	0100 0001 0101 1010 0101 1000 — 4 1 5 A 5 8	

Step 7:
Set the appropriate L, E, or G flag to reflect the result's relationship to zero.

Set G flag ON (sign of fraction positive)

After command execution:

B	101C	101D	101E	101F
(B)	0100 0010	0100 0001	0101 1010	0101 1000

Hexadecimal value = 42415A58

EXAMPLE 2 (Unlike Signs)

Assume that the fields used in the preceding example contained the following information:

A	0D44	0D45	0D46	0D47
(A)	0100 0010	0100 0001	0010 0100	0011 0110

Hexadecimal value = 42412436

B	101C	101D	101E	101F
(B)	1100 0000	0011 0110	0010 0010	0011 0100

Hexadecimal value = C0362234

	SIGN 0=+ 1=-	CHARACTERISTIC	FRACTION	GUARD DIGIT
(A)	0	100 0010 4 2	0100 0001 0010 0100 0011 0110 4 1 2 4 3 6	
(B)	1	100 0000 C 0	0011 0110 0010 0010 0011 0100 3 6 2 2 3 4	

Step 1:
Calculate the difference between the characteristics.

| | | 000 0010
0   2 | | |

Step 2:
The difference is less than 7. Shift the fraction with the smaller characteristic right the number of positions specified by the difference.

| (B) | | | 0011 0110 0010 0010 0011 0100
3   6   2   2   3   4 | |
| (B) | | | 0000 0000 0011 0110 0010 0010
0   0   3   6   2   2 | 0011
3 |

Step 3:
Compare signs. If unlike, next step is to subtract fractions. If like, next step is to add fractions.

(A)	0			
(B)	1			
	1		Unlike signs	

Step 4:
Subtract fractions to form intermediate sum. The larger of the two characteristics is subsequently referenced.

(A)			0100 0001 0010 0100 0011 0110 4 1 2 4 3 6	
(B)			0000 0000 0011 0110 0010 0010 0 0 3 6 2 2	0011 3
		100 0010 4 2	0100 0000 1110 1110 0001 0011 4 0 E E 1 3	1101 D

Step 5:
Normalize the intermediate sum and adjust the characteristic.

Already normalized

Step 6:
Truncate the result (drop right-most digit) and store in (B). The sign is governed by the rules of algebra.

| (B) | 0 | 100 0010
4   2 | 0100 0000 1110 1110 0001 0011
4   0   E   E   1   3 | |

Step 7:
Set the appropriate L, E, or G flag to reflect the result's relationship to zero.

Set G flag ON (sign of fraction positive)

After command execution:

B	101C	101D	101E	101F
(B)	0100 0010	0100 0000	1110 1110	0001 0011

Hexadecimal value = 4240EE13

EXAMPLE 3 (Characteristic difference of 7 or greater)

Assume that the fields used in the preceding example contained the following information:

A	0D44	0D45	0D46	0D47
(A)	0101 0010	0100 0001	0010 0100	0011 0110

Hexadecimal value = 52412436

B	101C	101D	101E	101F
(B)	0100 0000	0011 0110	0010 0010	0011 0100

Hexadecimal value = 40362234

	SIGN 0=+ 1=−	CHARACTERISTIC	FRACTION	GUARD DIGIT
(A)	0	101 0010 5 2	0100 0001 0010 0100 0011 0110 4 1 2 4 3 6	
(B)	0	100 0000 4 0	0011 0110 0010 0010 0011 0100 3 6 2 2 3 4	

Step 1:
Calculate the difference between the characteristics.

     001 0010   1   2

Step 2:
The difference is 7 or greater. The fraction with the larger characteristic automatically becomes the intermediate sum, and the larger characteristic is subsequently referenced. The guard digit is zero, no shift occurred.

     101 0010   5   2   0100 0001 0010 0100 0011 0110   4   1   2   4   3   6   0000   0

Step 3:
Normalize the intermediate sum and adjust the characteristic.

     Already normalized

Step 4:
Truncate the result (drop right-most digit) and store in (B). The sign is retained from the floating point number with the larger characteristic.

(B)   0   101 0010   5   2   0100 0001 0010 0100 0011 0110   4   1   2   4   3   6

Step 5:
Set the appropriate L, E, or G flag to reflect the result's relationship to zero.

     Set G flag ON (sign of fraction positive)

After command execution:

B (B)	101C	101D	101E	101F
	0101 0010	0100 0001	0010 0100	0011 0110

Hexadecimal value = 52412436

EXAMPLE 4 (Fraction Overflow)

Assume that the fields used in the preceding example contained the following information:

A	0D44	0D45	0D46	0D47
(A)	0100 0010	1111 1110	0010 0100	0011 0110

Hexadecimal value = 42FE2436

B	101C	101D	101E	101F
(B)	0100 0001	0011 0110	0010 0010	0011 0100

Hexadecimal value = 41362234

		CHARACTERISTIC		FRACTION					GUARD DIGIT

SIGN
0=+
1=-

	SIGN	CHARACTERISTIC	FRACTION	GUARD DIGIT
(A)	0	100 0010 / 4 2	1111 1110 0010 0100 0011 0110 / F E 2 4 3 6	
(B)	0	100 0001 / 4 1	0011 0110 0010 0010 0011 0100 / 3 6 2 2 3 4	

Step 1:
Calculate the difference between the characteristics.

	CHARACTERISTIC
	000 0001 / 0 1

Step 2:
The difference is less than 7. Shift the fraction with the smaller characteristic right the number of positions specified by the difference.

	FRACTION	GUARD DIGIT
(B)	0011 0110 0010 0010 0011 0100 / 3 6 2 2 3 4	
(B)	0000 0011 0110 0010 0010 0011 / 0 3 6 2 2 3	0100 / 4

Step 3:
Compare signs. If unlike, next step is to subtract fractions. If like, next step is to add fractions.

(A) 0
(B) 0
0   Like signs

Step 4:
Add fractions to form intermediate sum. The larger of the two characteristics is subsequently referenced.

	FRACTION	GUARD DIGIT
(A)	1111 1110 0010 0100 0011 0110 / F E 2 4 3 6	
(B)	0000 0011 0110 0010 0010 0011 / 0 3 6 2 2 3	0100 / 4

CHARACTERISTIC	FRACTION	GUARD DIGIT
100 0010 / 4 2	10000 0001 1000 0110 0101 1001 / 1 0 1 8 6 5 9	0100 / 4

Fraction overflow carry

Step 5:
An overflow carry occurred. The intermediate sum is shifted right one digit to make room for the carry. The characteristic is increased by one.

CHARACTERISTIC	FRACTION	GUARD DIGIT
000 0001 / 0 1		
100 0011 / 4 3	0001 0000 0001 1000 0110 0101 / 1 0 1 8 6 5	1001 / 9

Step 6:
Truncate the result (drop rightmost digit) and store in (B). The sign is governed by the rules of algebra.

	SIGN	CHARACTERISTIC	FRACTION
(B)	0	100 0011 / 4 3	0001 0000 0001 1000 0110 0101 / 1 0 1 8 6 5

Step 7:
Set the appropriate L, E, or G Flag to reflect the result's relationship to zero.

Set G Flag ON (sign of fraction positive)

After command execution:

B	101C	101D	101E	101F
(B)	0100 0011	0001 0000	0001 1000	0100 0101

Hexadecimal value = 43101865

EXAMPLE 5 (Normalization Required)

Assume that the fields used in the preceding example contained the following information:

A	0D44	0D45	0D46	0D47
(A)	0100 0010	0001 0000	0001 0010	0100 0011

Hexadecimal value = 42101243

B	101C	101D	101E	101F
(B)	1100 0001	0011 0110	0010 0010	0011 0100

Hexadecimal value = C1362234

	SIGN 0=+ 1=-	CHARACTERISTIC	FRACTION	GUARD DIGIT
(A)	0	100 0010 / 4 2	0001 0000 0001 0010 0100 0011 / 1 0 1 2 4 3	
(B)	1	100 0001 / C 1	0011 0110 0010 0010 0011 0100 / 3 . 6 2 2 3 4	

Step 1:
Calculate the difference between the characteristics.

		000 0001 / 0 1		

Step 2:
The difference is less than 7. Shift the fraction with the smaller characteristic right the number of positions specified by the difference.

			FRACTION	GUARD DIGIT
(B)			0011 0110 0010 0010 0011 0100 / 3 6 2 2 3 4	
(B)			0000 0011 0110 0010 0010 0011 / 0 3 6 2 2 3	0100 / 4

Step 3:
Compare signs. If unlike, next step is to subtract fractions. If like, next step is to add fractions.

(A)	0			
(B)	1			
	1		Unlike signs	

Step 4:
Subtract fractions to form intermediate sum. The larger of the two characteristics is subsequently referenced.

(A)			0001 0000 0001 0010 0100 0011 / 1 0 1 2 4 3	
(B)			0000 0011 0110 0010 0010 0011 / 0 3 6 2 2 3	0100 / 4
		100 0010 / 4 2	0000 1100 1011 0000 0001 1111 / 0 C B 0 1 F	1100 / C

Step 5:
Normalize the intermediate sum and adjust the characteristic by subtracting the number of left shifts required for the normalization.

		100 0010 / 4 2	1100 1011 0000 0001 1111 1100 / C B 0 1 F C	0000 / 0
		000 0001 / 0 1	= Number of shifts that were required for the normalization.	
		100 0001 / 4 1	1100 1011 0000 0001 1111 1100 / C B 0 1 F C	0000 / 0

Step 6:
Truncate the result (drop rightmost digit) and store in (B). The sign is governed by the rules of algebra.

| (B) | 0 | 100 0001 / 4 1 | 1100 1011 0000 0001 1111 1100 / C B 0 1 F C | |

Step 7:
Set the appropriate L, E, or G Flag to reflect the result's relationship to zero.

Set G flag ON (sign of fraction positive)

After command execution:

B	101C	101D	101E	101F
(B)	0100 0001	1100 1011	0000 0001	1111 1100

Hexadecimal value = 41CB01FC

EXAMPLE 6 (Characteristic Overflow)

Assume that the fields used in the preceding example contained the following information:

A	0D44	0D45	0D46	0D47
(A)	0111 1111	1111 1110	0010 0100	0011 0110

Hexadecimal value = 7FFE2436

B	101C	101D	101E	101F
(B)	0111 1111	0011 0110	0010 0010	0011 0100

Hexadecimal value = 7F362234

CHARACTERISTIC   GUARD
SIGN   FRACTION   DIGIT
0=+
1=-

	SIGN	CHARACTERISTIC	FRACTION	GUARD DIGIT
(A)	0	111 1111 / 7 F	1111 1110 0010 0100 0011 0110 / F E 2 4 3 6	
(B)	0	111 1111 / 7 F	0011 0110 0010 0010 0011 0100 / 3 6 2 2 3 4	

Step 1:
Calculate the difference between the characteristics.

000 0000 / 0 0

Step 2:
The difference is zero No shifting required

Step 3:
Compare the signs. If unlike, next step is to subtract fractions. If like, next step is to add fractions.

(A) 0
(B) 0
0 Like signs

Step 4:
Add fractions to form intermediate sum. No shift occurred; a guard digit of zero is assumed.

(A) 1111 1110 0010 0100 0011 0110 / F E 2 4 3 6
(B) 0011 0110 0010 0010 0011 0100 / 3 6 2 2 3 4

111 1111 / 7 F | 10011 0100 0100 0110 0110 1010 / 1 3 4 4 6 6 A | 0000 / 0

Fraction overflow carry

Step 5:
An overflow carry occurred. The intermediate sum is shifted right one digit to make room for the carry. The characteristic is increased by one.

000 0001 / 0 1

1 | 000 0000 / 0 0 | 0001 0011 0100 0100 0110 0110 / 1 3 4 4 6 6 | 1010 / A

Characteristic overflow carry

Step 6:
Check for characteristic overflow carry. If ON, set PE ON.

Set PE ON.

Step 7:
Truncate the result (drop rightmost digit) and store in (B). The sign is governed by the rules of algebra.

(B) 0 | 000 0000 / 0 0 | 0001 0011 0100 0100 0110 0110 / 1 3 4 4 6 6

Step 8:
Set the appropriate L, E, or G flag to reflect the result's relationship to zero.

Set G flag ON (sign of fraction positive)

Characteristic Invalid due to loss of overflow carry. PE trap taken.

After command execution:

B	101C	101D	101E	101F
(B)	0000 0000	0001 0011	0100 0100	0110 0110

Hexadecimal value = 00134466

EXAMPLE 7 (True Zero Generated)

Assume that the fields used in the preceding example contained the following information:

A	0D44	0D45	0D46	0D47
(A)	0110 0011	1111 1110	0010 0100	0011 0110

Hexadecimal value = 63FE2436

B	101C	101D	101E	101F
(B)	1110 0011	1111 1110	0010 0100	0011 0110

Hexadecimal value = E3FE2436

	SIGN 0=+ 1=-	CHARACTERISTIC	FRACTION	GUARD DIGIT

	SIGN	CHARACTERISTIC	FRACTION	GUARD DIGIT
(A)	0	110 0011 6 3	1111 1110 0010 0100 0011 0110 F E 2 4 3 6	
(B)	1	110 0011 E 3	1111 1110 0010 0100 0011 0110 F E 2 4 3 6	

Step 1:
Calculate the difference between the characteristics.

000 0000
0   0

Step 2:
The difference is zero.

No shifting required

Step 3:
Compare signs. If unlike, next step is to subtract fractions. If like, next step is to add fractions.

(A) 0
(B) 1

1   Unlike signs

Step 4:
Subtract fractions to form intermediate sum. No shift occurred; a guard digit of zero is assumed.

(A)   1111 1110 0010 0100 0011 0110
       F   E   2   4   3   6

(B)   1111 1110 0010 0100 0011 0110
       F   E   2   4   3   6

110 0011   0000 0000 0000 0000 0000 0000 | 0000
6   3   0   0   0   0   0   0 | 0

Step 5:
Check the intermediate sum of the fractions for being all zeros. If all zeros generate a true zero.

Intermediate sum — all zeros

0000 0000   0000 0000 0000 0000 0000 0000 | 0000
0   0   0   0   0   0   0   0 | 0

Step 6:
Truncate the result (drop right-most digit) and store in (B).

(B) 0000 0000   0000 0000 0000 0000 0000 0000
    0   0   0   0   0   0   0   0

Step 7:
Set the appropriate L, E, or G flag to reflect the result's relationship to zero.

Set E flag ON (true zero)

After command execution:

B (B)	101C	101D	101E	101F
	0000 0000	0000 0000	0000 0000	0000 0000

Hexadecimal value = 00000000

EXAMPLE 8 (Characteristic Underflow)

Assume that the fields used in the preceding example contained the following information:

A	0D44	0D45	0D46	0D47
(A)	0000 0000	0100 0001	0010 0100	0011 0110

Hexadecimal value = 00412436

B	101C	101D	101E	101F
(B)	1000 0000	0100 0000	0010 0010	0011 0100

Hexadecimal value = 80402234

	SIGN 0=+ 1=−	CHARACTERISTIC	FRACTION	GUARD DIGIT
(A)	0	000 0000 0 0	0100 0001 0010 0100 0011 0110 4 1 2 4 3 6	
(B)	1	000 0000 8 0	0100 0000 0010 0010 0011 0100 4 0 2 2 3 4	

Step 1:
Calculate the difference between the characteristics.

| | | 000 0000 0   0 | | |

Step 2:
The difference is zero.

No shifting required

Step 3:
Compare signs. If unlike, next step is to subtract fractions. If like, next step is to add fractions.

(A) 0
(B) 1
    1   Unlike signs

Step 4:
Subtract fractions to form intermediate sum. No shift occurred; a guard digit of zero is assumed.

(A)			0100 0001 0010 0100 0011 0110 4 1 2 4 3 6	
(B)			0100 0000 0010 0010 0011 0100 4 0 2 2 3 4	
		000 0000 0 0	0000 0001 0000 0010 0000 0010 0 1 0 2 0 2	0000 0

Step 5:
Normalize the intermediate sum and adjust the characteristic by subtracting the number of left shifts required for the normalization.

		000 0000 0 0	0001 0000 0010 0000 0010 0000 1 0 2 0 2 0	0000 0
		000 0001 0 1	= Number of shifts that were required for the normalization.	
		111 1111 7 F	0001 0000 0010 0000 0010 0000 1 0 2 0 2 0	0000 0

Complement of 000 0001.
Characteristic Underflow resulted from trying to subtract 0 1 from 0 0.

Step 6:
Characteristic underflow resulted. Set overflow indicator (OF) ON. Generate a true zero for the result.

Set overflow (OF) ON.

| | 0 | 000 0000 0   0 | 0000 0000 0000 0000 0000 0000
0   0   0   0   0   0 | 0000 0 |

Step 7:
Truncate the result (drop rightmost digit) and store the result in (B). Set the appropriate L,E, or G flag to reflect the result's relationship to zero.

| (B) | 0 | 000 0000 0   0 | 0000 0000 0000 0000 0000 0000
0   0   0   0   0   0 | |

Set E flag ON (true zero)

After command execution:

B (B)	101C	101D	101E	101F
	0000 0000	0000 0000	0000 0000	0000 0000

Hexadecimal value = 00000000

EXAMPLE 9 (Negative Result)

Assume that the fields used in the preceding example contained the following information:

A (A)	0D44	0D45	0D46	0D47
	1100 0010	0100 0111	0010 0100	0011 0110

Hexadecimal value = C2472436

B (B)	101C	101D	101E	101F
	0100 0010	0011 0110	0010 0010	0011 0100

Hexadecimal value = 42362234

	SIGN 0=+ 1=–	CHARACTERISTIC	FRACTION	GUARD DIGIT
(A)	1	100 0010 C 2	0100 0111 0010 0100 0011 0110 4 7 2 4 3 6	
(B)	0	100 0010 4 2	0011 0110 0010 0010 0011 0100 3 6 2 2 3 4	

Step 1:
Calculate the difference between the characteristics.

000 0000
0   0

Step 2:
The difference is zero.

No shifting required

Step 3:
Compare signs. If unlike, next step is to subtract fractions. If like, next step is to add fractions.

(A) 1
(B) 0
1

Unlike signs

Step 4:
Subtract fractions to form intermediate sum. No shift occurred; a guard digit of zero is assumed.

(A)   0100 0111 0010 0100 0011 0110
       4   7   2   4   3   6

(B)   0011 0110 0010 0010 0011 0100
       3   6   2   2   3   4

100 0010   0001 0001 0000 0010 0000 0010   0000
4   2   1   1   0   2   0   2   0

Step 5:
Normalize the intermediate sum and adjust the characteristic.

Already normalized

Step 6:
Truncate the result (drop rightmost digit) and store the result in (B). The sign is governed by the rules of algebra.

(B) 1 100 0010   0001 0001 0000 0010 0000 0010
      C   2   1   1   0   2   0   2

Step 7:
Set the appropriate L, E, or G flag to reflect the result's relationship to zero.

Set L flag ON (sign of fraction negative)

After command execution:

B	101C	101D	101E	101F
(B)	1100 0010	0001 0001	0000 0010	0000 0010

Hexadecimal value = C2110202

C = 117(75)(F5)

The contents of the field specified by the effective A operand address are added to the contents of the field specified by the effective B operand address. Each field is considered to contain a floating point, double precision number (2-digit sign/characteristic and 14-digit fraction); addition is performed on the fractions. The result of the addition is an intermediate sum which is normalized to give a final sum. The final sum replaces (B).

In performing the operation, the characteristics of the two operands are compared. If the difference is less than 14, the fraction with the smaller characteristic is right shifted the number of hexadecimal digits specified by that difference. The fractions are then added algebraically to form the intermediate sum. This includes checking of signs and complementary addition (subtraction) if unlike signs. The larger of the two characteristics is subsequently referenced. If a fraction overflow carry results from the addition, the intermediate sum is right shifted one digit to make room for the carry, and the characteristic is increased by one. If this increase causes a characteristic overflow, a PE trap occurs with the erroneous result stored in (B).

If the difference between the characteristics of the two operands is 14 or greater, the fraction with the larger characteristic automatically becomes the intermediate sum. If the fraction with the smaller characteristic had been shifted to the right 14 positions, zeros would have been added to the fraction with the larger characteristic giving the same result. The larger characteristic is subsequently referenced.

The intermediate sum of the fractions consists of 14 hexadecimal digits, including the possible overflow carry. If these digits are all zero, a true zero (64 zero bits) is generated for the final result (sign, characteristic, and fraction all zeros).

If a true zero is not generated, the intermediate sum of the fractions is left-shifted as necessary to form a normalized fraction. Vacated low order (right-most) digit positions are filled with zeros and the characteristic is reduced by the amount of the shift. If this normalization causes the characteristic to underflow, a true zero is generated (64 zero bits), and the overflow indicator (OF) is set ON.

Finally, the intermediate sum with its characteristic (64 bits altogether) is stored in (B).

The sign of the sum is derived by the rules of algebra. A zero fraction cannot result, unless a true zero results or is generated, in which case the sign is always positive.

One of the Less, Equal, or Greater flags is set ON, reflecting the results: less than, equal to, or greater than zero.

T is not used unless this command is to be followed by a 1-address command. In this case the T specified would be retained for use in the following command, but would have no effect on this command. The length of B after execution is equal to the length of B before execution.

Command Execution Time

$$E = 2\ PO + 20P + GP + K\ (2\ PO + 14P) + X\ (2\ PO + 15P + \frac{1 + N}{2}P)$$

$$+ Y\ (2\ PO + 16P + \frac{M}{2}P)$$

The constants G, K, X, N, Y, and M are defined the same as F.P. ADD SINGLE.

EXAMPLE 1 (Like Signs)
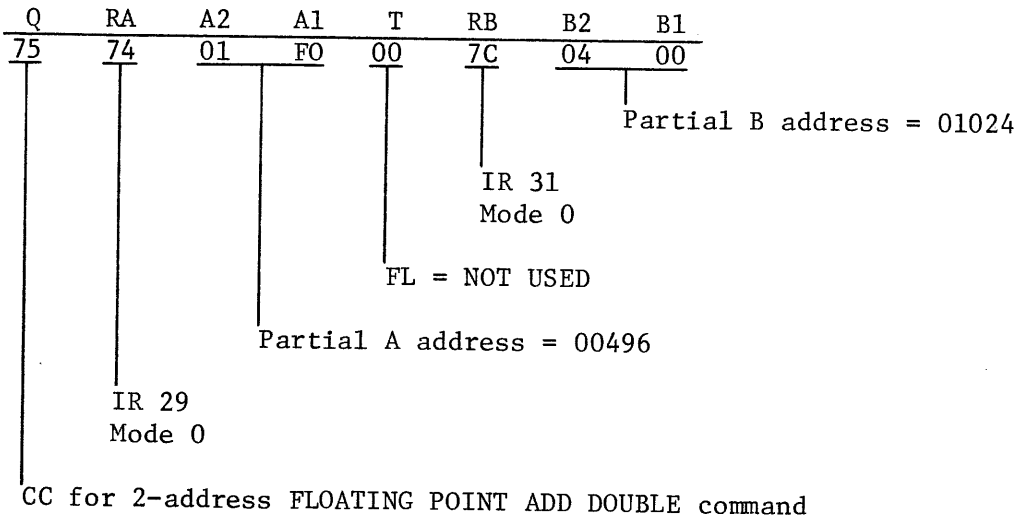
```
     Q    RA    A2    A1    T    RB    B2    B1
    75    74    01    FO    00   7C    04    00
                                              |
                                              └─ Partial B address = 01024

                                        IR 31
                                        Mode 0

                             FL = NOT USED

                   Partial A address = 00496

         IR 29
         Mode 0

CC for 2-address FLOATING POINT ADD DOUBLE command
```

Assume the index registers contain the following:

(IR 29) = OB54 (02900)
(IR 31) = OC1C (03100)

After command setup:

Effective A address = OB54 + 01FO = OD44 (03396)
Effective B address = OC1C + 0400 = 101C (04124)

Before command execution:

A	0D44	0D45	0D46	0D47	0D48	0D49	0D4A	0D4B
(A)	0100 0101	0001 0010	0011 0100	0101 0110	0111 1000	1001 1010	1011 1100	1101 1110

Hexadecimal value = 45123456789ABCDE

B	101C	101D	101E	101F	1020	1021	1022	1023
(B)	0100 0000	0001 0010	0011 0100	0101 0110	0111 1000	1001 1010	1011 1100	1101 1110

Hexadecimal value = 40123456789ABCDE

Step 1:
Calculate the difference between the characteristics.
   (a) Difference = 14 or greater.
       The fraction with the larger characteristic automatically becomes the
       intermediate sum. Bypass steps 3, 4, and 5.
   (b) Difference = less than 14.
       The fraction with the smaller characteristic is right-shifted the
       number of positions indicated by the difference. There is no guard
       digit position, therefore, any character right-shifted from the least
       significant (rightmost) position of the fraction is lost.

Step 2:
Compare signs. If unlike, next step is to subtract fractions. If like, next
step is to add fractions.

Step 3:
Add or subtract fractions (depending on the results of Step 2) to form the
intermediate sum. The larger characteristic is subsequently referenced.

Step 4:
If step 3 caused fraction overflow, right-shift the intermediate sum one digit
position to make room for the carry and increase the characteristic by one.

Step 5:
If step 4 caused characteristic overflow, set the PE indicator ON.

Step 6:
Check the intermediate sum for all zeros. If all zeros, generate a true zero
(64 bits).

Step 7:
If a true zero was not generated, normalize the intermediate sum and reduce
the characteristic by the amount of digit positions shifted.

Step 8:
If step 7 caused characteristic underflow, generate a true zero and set the
overflow indicator (OF) ON.

Step 9:
Store the result in (B). The sign is governed by the rules of algebra.

Step 10:
Set the appropriate L, E, or G flag to indicate the result's relationship
to zero.

```
SIGN ┐    CHARACTERISTIC
0=+  │    │                                              FRACTIONS
1=-  ▼    ▼
```

(A)   0   100 0101   0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110
           4   5   1   2   3   4   5   6   7   8   9   A   B   C   D   E

(B)   0   100 0000   0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110
           4   0   1   2   3   4   5   6   7   8   9   A   B   C   D   E

Step 1:   000 0101
           0   5

Step 1b:
(B)   0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110
                       1   2   3   4   5   6   7   8   9   A   B   C   D   E

(B)   0000 0000 0000 0000 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001
                       0   0   0   0   0   1   2   3   4   5   6   7   8   9

Step 2:
(A)   0

(B)   0

      0   Like Signs

Step 3:
(A)   0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110
                       1   2   3   4   5   6   7   8   9   A   B   C   D   E

(B)   0000 0000 0000 0000 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001
                       0   0   0   0   0   1   2   3   4   5   6   7   8   9

      100 0101   0001 0010 0011 0100 0101 0111 1001 1011 1110 0000 0010 0100 0110 0111
       4   5   1   2   3   4   5   7   9   B   E   0   2   4   6   7

Step 4:   Bypass Step 4 (No fraction overflow)

Step 5:   Bypass Step 5 (No characteristic overflow)

Step 6:   Intermediate Sum NOT all zeros.

Step 7:   Intermediate Sum already normalized.

Step 8:   Bypass Step 8 (No characteristic underflow)

Step 9:
(B)   0   100 0101   0001 0010 0011 0100 0101 0111 1001 1011 1110 0000 0010 0100 0110 0111
           4   5   1   2   3   4   5   7   9   B   E   0   2   4   6   7

Step 10:   Set G flag ON (Sign of fraction positive).

After command execution:

B (B)	101C	101D	101E	101F	1020	1021	1022	1023
	0100 0101	0001 0010	0011 0100	0101 0111	1001 1011	1110 0000	0010 0100	0110 0111

Hexadecimal value = 451234579BE02467

EXAMPLE 2 (Unlike signs)

Assume that the fields used in the preceding example contained the following
information:

A	0D44	0D45	0D46	0D47	0D48	0D49	0D4A	0D4B
(A)	0100 0010	0111 0110	1000 0100	0001 1010	0010 1101	0100 0100	0011 0100	0100 0100

Hexadecimal value = 4276841A2D443444

B	101C	101D	101E	101F	1020	1021	1022	1023
(B)	1100 0010	0111 0110	0111 0011	0100 0100	0100 0011	1010 0010	1101 1101	0011 0010

Hexadecimal value = C27634443A2DD32

(A)

(B)

Step 1:
Calculate the difference between the characteristics.

    (a) Difference = 14 or greater
        The fraction with the larger characteristic automatically becomes the
        intermediate sum. Bypass steps 3, 4, and 5.
    (b) Difference = less than 14
        The fraction with the smaller characteristic is right-shifted the num-
        ber of positions indicated by the difference. There is no guard digit
        position, therefore, any character right-shifted from the least signifi-
        cant (rightmost) position of the fraction is lost.

Step 2:
Compare signs. If unlike, next step is to subtract fractions. If like, next
step is to add fractions.

Step 3:
Add or subtract fractions (depending on the results of step 2) to form the
intermediate sum. The larger characteristic is subsequently referenced.

Step 4:
If step 3 caused fraction overflow, right-shift the intermediate sum one digit
position to make room for the carry and increase the characteristic by one.

Step 5:
If step 4 caused characteristic overflow, set the PE indicator ON.

Step 6:
Check the intermediate sum for all zeros. If all zeros, generate a true zero
(64 bits).

Step 7:
If a true zero was not generated, normalize the intermediate sum and reduce the
characteristic by the amount of digit positions shifted.

Step 8:
If step 7 caused characteristic underflow, generate a true zero and set the
overflow indicator (OF) ON.

Step 9:
Store the result in (B). The sign is governed by the rules of algebra.

Step 10:
Set the appropriate L, E, or G flag to indicate the result's relationship to
zero.

	SIGN 0=+ 1=-	CHARACTERISTIC	FRACTIONS

```
        SIGN┐ CHARACTERISTIC                    FRACTIONS
        0=+ │    ↓                                 ↓
        1=- ↓
(A)      │ 0 │ 100 0010  │ 0111 0110 1000 0100 0001 1010 0010 1101 0100 0100 0011 0100 0100 0100
         │   │   4    2  │   7    6    8    4    1    A    2    D    4    4    3    4    4    4

(B)      │ 1 │ 100 0010  │ 0111 0110 0111 0011 0100 0100 0100 0011 1010 0010 1101 1101 0011 0010
         │   │   C    2  │   7    6    7    3    4    4    4    3    A    2    D    D    3    2

Step 1:      │ 000 0000
             │   0    0

Step 1b:                   No shifting required (The difference is zero)

Step 2:
   (A)   │ 0 │
   (B)   │ 1 │
         │ 1 │             Unlike signs

Step 3:                    0111 0110 1000 0100 0001 1010 0010 1101 0100 0100 0011 0100 0100 0100
                             7    6    8    4    1    A    2    D    4    4    3    4    4    4

                           0111 0110 0111 0011 0100 0100 0100 0011 1010 0010 1101 1101 0011 0010
                             7    6    7    3    4    4    4    3    A    2    D    D    3    2

             │ 100 0010  │ 0000 0000 0001 0000 1101 0101 1110 1001 1010 0001 0101 0111 0001 0010
             │   4    2  │   0    0    1    0    D    5    E    9    A    1    5    7    1    2

Step 4:                    Bypass Step 4 (No fraction overflow)
Step 5:                    Bypass Step 5 (No characteristic overflow)
Step 6:                    Intermediate sum NOT all zeros.

Step 7:      │ 100 0010  │ 0001 0000 1101 0101 1110 1001 1010 0001 0101 0111 0001 0010 0000 0000
             │   4    2  │   1    0    D    5    E    9    A    1    5    7    1    2    0    0

             │ 000 0010  = Number of shifts that were required for the normalization
             │   0    2

             │ 100 0000  │ 0001 0000 1101 0101 1110 1001 1010 0001 0101 0111 0001 0010 0000 0000
             │   4    0  │   1    0    D    5    E    9    A    1    5    7    1    2    0    0

Step 8:                    Bypass Step 8 (No characteristic underflow)

Step 9:  │ 0 │ 100 0000  │ 0001 0000 1101 0101 1110 1001 1010 0001 0101 0111 0001 0010 0000 0000
   (B)   │   │   4    0  │   1    0    D    5    E    9    A    1    5    7    1    2    0    0

Step 10                    Set G flag ON (sign of fraction positive)
```

After command execution:

B	101C	101D	101E	101F	1020	1021	1022	1023
(B)	0100 0000	0001 0000	1101 0101	1110 1001	1010 0001	0101 0111	0001 0010	0000 0000

Hexadecimal value = 4010D5E9A1571200

C = 118(76)(F6)

General

This command subtracts the contents of the field specified by the effective A address from the field specified by the effective B address, and the normalized result replaces the contents of the B field, (B)-(A)=(B)

The contents of the A and B fields are considered to be in floating point, single-precision format.

Subtraction of the fields takes place in three basic stages: the sign of the A field is inverted, the characteristics are compared, and the fractions are added. During the execution of the command an intermediate sum is generated. Although the fractions of the A and B fields are 6 hexadecimal digit fractions, the intermediate sum may be expressed as a 7 hexadecimal digit fraction. The seventh (rightmost) digit is called a guard digit and is used to increase the precision of the final result. The guard digit is stored in a hardware register and not in a memory location. Therefore, upon termination of the command, the final result stored in the area specified by the B operand is a normalized 6-digit fraction.

Functional

1. Invert the sign of the A operand.
2. Compare the characteristics of the A and B operands.

   - If the difference is less than 7, the fraction with the smaller characteristic is right-shifted the number of hexadecimal digit positions specified by that difference. (This makes both the characteristics equal to the value of the largest original characteristic.) The fractional fields are then added algebraically to form an intermediate sum; if an overflow carry occurs, the intermediate sum is right-shifted one digit position to make room for the carry and the characteristic is increased by one. If this increase causes a characteristic overflow, a PE trap occurs with the overflow flag OFF. If the intermediate sum is all zeros, a true zero will be generated.
   - If the difference is 7 or greater, the larger number becomes the intermediate sum, with a guard digit of zero.

3. The intermediate sum, after possible re-complementing, consists of seven digits, the first six are the result (including overflow carry) and the seventh is the guard digit. If all the digits are zero, a true zero (32 zero bits) is generated as the intermediate sum.
4. If a true zero is not generated, the intermediate sum including guard digit is left-shifted as necessary to form a normalized fraction; vacated low-order digit positions are filled with zeros and the characteristic is reduced by the amount of the shift.

5. If the normalization causes characteristic underflow, a true zero is generated and the overflow indicator is turned ON; no PE trap occurs.

6. In any case, the intermediate result is truncated to the proper length and the result is then stored as the final result in the area specified by the B operand. The sign of the result is derived by the rules of algebra. A zero result cannot occur unless a true zero results or is generated, in which case the sign is always positive. One of the LEG flags is turned ON to indicate the relation of the result to zero.

T is not used. However, it must be equal to a specific configuration if the subsequent command is a 1-address command.

$$\mathbf{A} = B$$

Command Execution Time

$$E = 2\ PO + 12P + GP + K\ (2\ PO + 8P) + C\ (2\ PO + 8P)$$

$$+ X\ (2\ PO + 9P + \frac{1 + N}{2}P) + Y\ (\ 2\ PO + 10P + \frac{M}{2}P)$$

G, K, X, N, Y, and M are the same as F.P. ADD SINGLE.

C = 1, If the result of a subtraction produces an intermediate result in complement form.

HEXADECIMAL SUBTRACTION TABLE																
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
1	bF	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E
2	bE	bF	0	1	2	3	4	5	6	7	8	9	A	B	C	D
3	bD	bE	bF	0	1	2	3	4	5	6	7	8	9	A	B	C
4	bC	bD	bE	bF	0	1	2	3	4	5	6	7	8	9	A	B
5	bB	bC	bD	bE	bF	0	1	2	3	4	5	6	7	8	9	A
6	bA	bB	bC	bD	bE	bF	0	1	2	3	4	5	6	7	8	9
7	b9	bA	bB	bC	bD	bE	bF	0	1	2	3	4	5	6	7	8
8	b8	b9	bA	bB	bC	bD	bE	bF	0	1	2	3	4	5	6	7
9	b7	b8	b9	bA	bB	bC	bD	bE	bF	0	1	2	3	4	5	6
A	b6	b7	b8	b9	bA	bB	bC	bD	bE	bF	0	1	2	3	4	5
B	b5	b6	b7	b8	b9	bA	bB	bC	bD	bE	bF	0	1	2	3	4
C	b4	b5	b6	b7	b8	b9	bA	bB	bC	bD	bE	bF	0	1	2	3
D	b3	b4	b5	b6	b7	b8	b9	bA	bB	bC	bD	bE	bF	0	1	2
E	b2	b3	b4	b5	b6	b7	b8	b9	bA	bB	bC	bD	bE	bF	0	1
F	b1	b2	b3	b4	b5	b6	b7	b8	b9	bA	bB	bC	bD	bE	bF	0

(B)

(A)

b = Borrow from preceding digit

The following command will be used for all of the examples of FLOATING POINT
SUBTRACT SINGLE. Only the contents of A and B will vary to change the examples.

EXAMPLE 1

Q	RA	A2	A1	T	RB	B2	B1
76	74	01	F0	00	7C	04	00

Partial B address = 01024

IR 31
Mode 0

FL = NOT USED

Partial A address = 00496

IR 29
Mode 0

CC for FLOATING POINT SUBTRACT SINGLE

Assume the index registers contain the following:

    IR 29 = 0B54 (02900 decimal)
    IR 31 = 0C1C (03100 decimal)

After command setup:

Effective A address = 0B54 + 01F0 = 0D44 (03396)
Effective B address = 0C1C + 0400 = 101C (04124)

Before command execution:

A (A)	0D44	0D45	0D46	0D47
	0100 0000	1001 1000	0111 0110	0101 0100

40987654

B (B)	101C	101D	101E	101F
	0100 0001	1001 1000	0111 0110	0101 0100

41987654

	SIGN	CHARACTERISTIC	FRACTION	GUARD DIGIT
(B)	0	100 0001	1001 1000 0111 0110 0101 0100	
		4 1	9 8 7 6 5 4	
(A)	0	100 0000	1001 1000 0111 0110 0101 0100	
		4 0	9 8 7 6 5 4	

Step 1:
Invert the sign of (A)

	SIGN	CHARACTERISTIC	FRACTION	GUARD DIGIT
(A)	1	100 0000	1001 1000 0111 0110 0101 0100	
		C 0	9 8 7 6 5 4	

Step 2:
Calculate the difference between the characteristics.

	CHARACTERISTIC
(B)	100 0001
(A)	100 0000
Difference	000 0001

Step 3:
Difference is less than 7; shift the smaller field (A or B) to the right,(the number of shifts is the difference as calculated in step 2.)

	FRACTION	GUARD DIGIT
(A)	1001 1000 0111 0110 0101 0100	
(A)	0000 1001 1000 0111 0110 0101	0100
	0 9 8 7 6 5	4

Step 4:
Compare signs of the characteristics; if like add, if unlike subtract.

	SIGN	FRACTION	GUARD DIGIT
(B)	0	1001 1000 0111 0110 0101 0100	
(A)	1	0000 1001 1000 0111 0110 0101	0100
	unlike		
Intermediate sum		1000 1110 1110 1110 1110 1110	1100
		8 E E E E E	C

Step 5 :
Check intermediate sum for all zeros. If all zeros, generate true zeros.

Bypass

Step 6 :
Normalize the intermediate sum and adjust the characteristic, if not a true zero.

Bypass—already normalized.

Step 7 :
Check if adjusting characteristic caused characteristic underflow. If yes, set the overflow indicator.

Bypass—no adjusting done

Step 8 :
Truncate the intermediate sum (drop the guard digit)
Store the result in (B)

	SIGN	CHARACTERISTIC	FRACTION	GUARD DIGIT
	0	100 0001	1000 1110 1110 1110 1110 1110	1100
	0	100 0001	1000 1110 1110 1110 1110 1110	
		4 1	8 E E E E E	

Set the appropriate LEG flag

Set the G flag.

NOTE

This example illustrates subtraction of two floating point numbers whose signs are alike and characteristics are different.

After command execution:

The contents of A remain unchanged, and the contents of B are as indicated.

B	101C	101D	101E	101F
(B)	0100 0001	1000 1110	1110 1110	1110 1110

Hexadecimal value =
418EEEEE

EXAMPLE 2

Before command execution:

A	0D44	0D45	0D46	0D47
(A)	0111 1111	1001 1001	1001 1001	1001 1001

Hexadecimal value =
7F999999

B	101C	101D	101E	101F
(B)	1111 1111	1001 1001	1001 1001	1001 1001

Hexadecimal value =
FF999999

	SIGN	CHARACTERISTIC		FRACTION						GUARD DIGIT

Layout table:

```
                                         CHARACTERISTIC                          GUARD
                             SIGN┐    ↓              FRACTION                     DIGIT

                    (B)  1|111 1111| 1001 1001 1001 1001 1001 1001
                           F    F      9    9    9    9    9    9

                    (A)  0|111 1111| 1001 1001 1001 1001 1001 1001
                           7    F      9    9    9    9    9    9
Step 1:
Invert the sign of (A)   (A)  1|111 1111| 1001 1001 1001 1001 1001 1001
                                 F    F      9    9    9    9    9    9

Step 2:
Calculate the difference between  (B)  1|111 1111|
the characteristics.              (A)  1|111 1111|
               Difference              |000 0000|
Step 3:
If the difference is less than 7,
shift the smaller fraction to the          Difference equals zero, there-
right.(Number of shifts equal the          fore shift zero positions.
difference as calculated in step 2)

Step 4:
Compare signs of characteristics; (B) 1 ⎫        1001 1001 1001 1001 1001 1001
if alike add, if unlike complement (A) 1 ⎬ like   1001 1001 1001 1001 1001 1001
(A) and add.                             ⎭       10011 0011 0011 0011 0011 0010
               Intermediate sum                   1 3    3    3    3    3    2
          with fractional overflow
Step 5:                                   carry
Examine for fractional overflow carry            0001 0011 0011 0011 0011 0011
in the intermediate result; if on,                 1    3    3    3    3    3
shift the intermediate result right      |111 1111|
one position, insert the carry, and            1
add one to the characteristic.          1|000 0000|
          Characteristic overflow-a carry generated from bit 7 to 8
Step 6:
Check for characteristic overflow;               Set PE due to overflow.
if on, PE

Step 7:                                 1|000 0000| 0001 0011 0011 0011 0011 0011
Store the intermediate sum.              |8    0|     1    3    3    3    3    3
```

NOTE

This example illustrates how fractional overflow can occur and how it can **cause** characteristic overflow. The command terminates with a PE and the result stored in (B). The result stored in (B) is invalid; the sign is correct, the characteristic is incorrect and the fraction is correct.

After command execution:

B (B)	101C	101D	101E	101F
	1000 0000	0001 0011	0011 0011	0011 0011

Hexadecimal value =
80133333

EXAMPLE 3

Before command execution:

A	0D44	0D45	0D46	0D47
(A)	0100 0001	1001 1000	0111 0110	0101 0100

Hexadecimal value =
    41987654

B	101C	101D	101E	101F
(B)	0100 0001	1001 1000	0111 0110	0101 0100

Hexadecimal value =
    41987654

		SIGN	CHARACTERISTIC	FRACTION	GUARD DIGIT
	(B)	0	100 0001	1001 1000 0111 0110 0101 0100	
			4 1	9 8 7 6 5 4	
	(A)	0	100 0001	1001 1000 0111 0110 0101 0100	
			4 1	9 8 7 6 5 4	

Step 1:
Invert the sign of (A)

	(A)	1	100 0001	1001 1000 0111 0110 0101 0100	
			C 1	9 8 7 6 5 4	

Step 2:
Calculate the difference between (B) | 100 0001
the characteristics. (A) | 100 0001
Difference | 000 0000

Step 3:
Shift the smaller fraction right by
the difference calculated in step 2.

Difference is zero; no shift
necessary; assign zero guard | 0000
digit.

Step 4:
Compare the signs of the charac- (B) 0 ⎫
teristics; if alike add, if unlike (A) 1 ⎬ unlike
subtract.

				1001 1000 0111 0110 0101 0100	0000
				1001 1000 0111 0110 0101 0100	
Intermediate sum				0000 0000 0000 0000 0000 0000	0000
				0 0 0 0 0 0	0

Step 5:
Check for all zero intermediate sum,
if all zero, generate true zero.

		0	000 0000	0000 0000 0000 0000 0000 0000	0000
			0 0	0 0 0 0 0 0	0

Step 6:
Normalize intermediate sum, if not
true zero.

Bypass, because true zero.

Step 7:
Check for underflow (characteristic).
If underflow generate true zero and
turn on overflow indicator.

Bypass. This step applies
only if step 6 is used.

Step 8:
Truncate intermediate sum
 (drop guard digit)
Store the result in (B)

		0	000 0000	0000 0000 0000 0000 0000 0000	0000
		0	000 0000	0000 0000 0000 0000 0000 0000	
			0 0	0 0 0 0 0 0	

Set the appropriate flag L,E,G

Set the E flag

NOTE

This example shows the method of subtracting equal fields
which result in the creation of a true zero. The signifi-
cant operation is the detection of a zero intermediate sum
which causes the characteristic to be set to zero and the
sign of the result to be positive.

B	101C	101D	101E	101F
(B)	0000 0000	0000 0000	0000 0000	0000 0000

EXAMPLE 4

Before command execution:

A	0D44	0D45	0D46	0D47
(A)	0100 0000	0001 0001	0001 0001	0001 0000

Hexadecimal value =
        40111110

B	101C	101D	101E	101F
(B)	0100 0000	0001 0001	0001 0001	0001 0001

Hexadecimal value =
        40111111

		SIGN	CHARACTERISTIC	FRACTION	GUARD DIGIT
	(B)	0	100 0000 4 0	0001 0001 0001 0001 0001 0001 1 1 1 1 1 1	
	(A)	0	100 0000 4 0	0001 0001 0001 0001 0001 0000 1 1 1 1 1 0	
Step 1: Invert the sign of (A)	(A)	1	100 0000 C 0		
Step 2: Calculate the difference between the characteristics.	(B) (A)		100 0000 100 0000		
Difference			000 0000		
Step 3: Shift the smaller value right, if the difference is less than 7.				Difference is zero —no shifting necessary.	
Step 4: Compare the signs of the charac- teristics, if alike add, if unlike subtract. Intermediate sum	(B) (A)	0 } unlike 1 }		0001 0001 0001 0001 0001 0001 0001 0001 0001 0001 0001 0000 0000 0000 0000 0000 0000 0001	 0000 0000
Step 5: Examine for fractional overflow carry in the intermediate sum; if on, shift the sum right one position, insert the carry and add one to the characteristic.				Bypass—no fractional overflow.	
Step 6: Check intermediate sum for all zeros. If all zeros generate true zero.				Bypass—not all zeros.	
Step 7: Normalize the intermediate sum and adjust the characteristic if not true zero.				0000 0000 0000 0000 0000 0001 0000 0000 0000 0000 0000 0000	0000 0000
number of shifts			100 0000 0101		
adjusted characteristics		0	11 1011 3 B		
Step 8: Check whether adjusting characteristic caused characteristic underflow. If yes, set the overflow indicator.				Adjusting did not cause characteristic underflow	
Step 9: Truncate the intermediate sum (drop the guard digit)				0001 0000 0000 0000 0000 0000 0001 0000 0000 0000 0000 0000	0000

	SIGN	CHARACTERISTIC	FRACTION	GUARD DIGIT

Store the intermediate sum in (B)

```
        0011 1011   0001 0000 0000 0000 0000 0000
          3    B      1    0    0    0    0    0
```

Set the appropriate LEG flag.   Set the G flag.
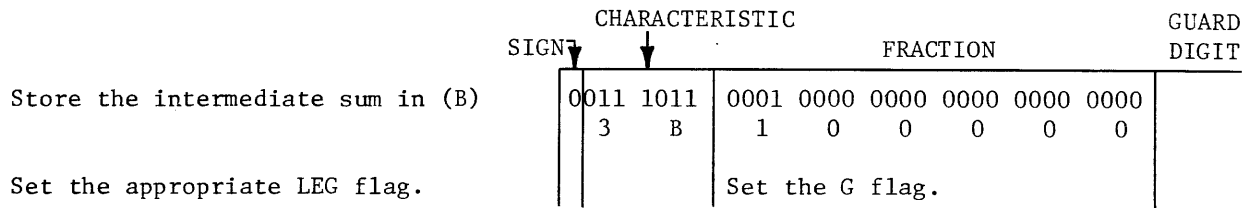
<u>NOTE</u>

This example shows the normal subtraction process of two floating point numbers whose signs and characteristics are alike. The significant section is the normalization of the intermediate sum (Step 7).

After command execution:

B	101C	101D	101E	101F
(B)	0011 1011	0001 0000	0000 0000	0000 0000

Hexadecimal value = 3B100000

EXAMPLE 5

Before command execution:

A	0D44	0D45	0D46	0D47
(A)	0100 0111	0001 0010	0011 0100	0101 0110

Hexadecimal value = 47123456

B	101C	101D	101E	101F
(B̄)	0100 0000	1001 1000	0111 0110	0101 0100

Hexadecimal value = 40987654

	SIGN	CHARACTERISTIC	FRACTION	GUARD DIGIT
(B)	0	100 0000 4 0	1001 1000 0111 0110 0101 0100 9 8 7 6 5 4	
(A)	0	100 0111 4 7	0001 0010 0011 0100 0101 0110 1 2 3 4 5 6	

Step 1:
Invert the sign of (A)

| (A) | 1 | 100 0111
C   7 | 0001 0010 0011 0100 0101 0110
1   2   3   4   5   6 | |

Step 2:
Calculate the difference between
the characteristics.

(B)		100 0000		
(A)		100 0111		
Difference		000 0111		

Step 3:
Difference is 7 or greater; the
field with the highest value is   (A)
taken as the intermediate sum,
and a guard digit of zero is
assigned.

| (A) | | | 0001 0010 0011 0100 0101 0110
1   2   3   4   5   6 | 0000
0 |

Step 4:
Check intermediate sum for all zeros,
and if all zeros generate a true zero.

Sum not equal to zero

Step 5:
Normalize the intermediate sum and
adjust the characteristic accordingly
(IF NOT TRUE ZERO)

Already normalized

Step 6:
Check the characteristic and see if step
5 caused underflow. If yes, set overflow
indicator.

No underflow

Step 7:
Truncate the intermediate sum
 (drop the guard digit)

| | | | 0001 0010 0011 0100 0101 0110
0001 0010 0011 0100 0101 0110
1   2   3   4   5   6 | 0000 |

Step 8:
Store the result in (B). The sign
is governed by the rules of alge-
bra. The characteristic is either
the result of step 4 or 5, or is
the higher original characteristic
of A or B.

| | 1 | 100 0111
C   7 | 0001 0010 0011 0100 0101 0110
1   2   3   4   5   6 | |

Set the appropriate LEG flag.

Set the L flag.

NOTE

This example shows the result when the difference in characteristics is 7 or greater. Since (A) is larger than (B), the result is a negative number. The intermediate sum under these conditions is the field with the higher value because an attempt to shift the lesser field to align the characteristics would have shifted the field so far to the right that it would not have participated in the addition. Therefore, as a means of saving time without loss of accuracy, the larger field is used as the intermediate result.

After command execution:

B	101C	101D	101E	101F
(B)	1100 0111	0001 0010	0011 0100	0101 0110

Hexadecimal value =
C7123456

EXAMPLE 6

Before command execution:

A	0D44	0D45	0D46	0D47
(A)	0000 0000	0000 0000	0001 0001	0001 0001

Hexadecimal value =
00001111

B	101C	101D	101E	101F
(B)	0000 0000	0000 0000	1001 1001	1001 1001

Hexadecimal value =
00009999

		SIGN	CHARACTERISTIC	FRACTION	GUARD DIGIT
	(B)	0	000 0000 0 0	0000 0000 1001 1001 1001 1001 0 0 9 9 9 9	
	(A)	0	000 0000 0 0	0000 0000 0001 0001 0001 0001 0 0 1 1 1 1	
Step 1: Invert the sign of (A)	(A)	1	000 0000 8 0	0000 0000 0001 0001 0001 0001 0 0 1 1 1 1	
Step 2: Calculate the difference between the characteristics. Difference	(B) (A)		000 0000 000 0000 000 0000		
Step 3: Difference less than 7; shift the smaller field right. Insert guard digit.				Characteristics equal; no shift	0000
Step 4: Compare the signs; if like add, if unlike subtract. Intermediate sum	(B) (A)	0 1	}unlike	0000 0000 1001 1001 1001 1001 0000 0000 0001 0001 0001 0001 0000 0000 1000 1000 1000 1000 0 0 8 8 8 8	0000 0000 0
Step 5: Check intermediate sum for all zeros. If all zeros generate true zero.				Bypass	
Step 6: Normalize the intermediate result and adjust the characteristic, if not true zero. characteristic (subtract) number of shifts			000 0000 000 0000 000 0010	0000 0000 1000 1000 1000 1000 1000 1000 1000 1000 0000 0000 8 8 8 8 0 0	0000 0000 0
				Since the result will be a value less than zero the underflow condition is true.	
Step 7: Check for underflow of characteristic. If true, set the overflow indicator and generate true zero.		0	000 0000	Set the overflow indicator. 0000 0000 0000 0000 0000 0000	0000
Step 8: Truncate the intermediate sum (drop the guard digit)		0 0	000 0000 000 0000	0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000	0000
Store the result in (B). The sign is determined by rules of algebra.		0	0 0	0 0 0 0 0 0	
Set the appropriate LEG flag.				Set the E flag.	

After command execution:

B	101C	101D	101E	101F
(B)	0000 0000	0000 0000	0000 0000	0000 0000

Hexadecimal value =
00000000

Overflow indicator ON.

C = 119(77)(F7)

General

The FLOATING POINT SUBTRACT DOUBLE command functions the same as FLOATING POINT SUBTRACT SINGLE, except that there is not a guard digit in the intermediate sum and the fields are double precision. The following examples illustrate the command flow.

Command Execution Time

$$E = 2PO + 20P + GP + K\ (2PO + 14P) + C\ (2PO + 14P)$$

$$+ X\ (2\ PO + 15P + \frac{1 + N}{2}P) + Y\ (2\ PO + 16P + \frac{M}{2}P)$$

The constants G, K, C, X, N, Y, and M are defined the same as F.P. ADD SINGLE.

The following command will be used for all the examples of F.P. SUBTRACT DOUBLE, only the contents of A and B will vary.
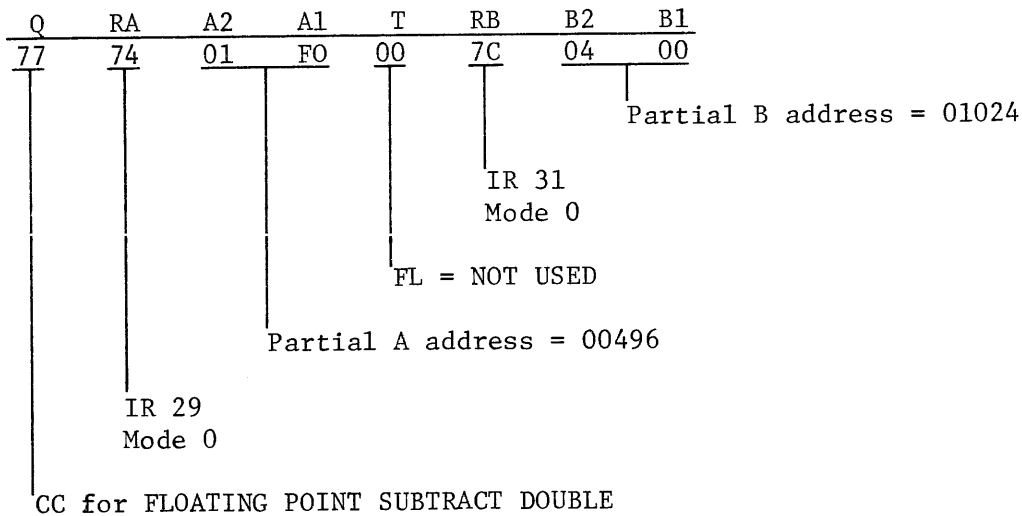
```
 Q    RA    A2    A1    T    RB    B2    B1
 77   74    01    F0    00   7C    04    00
```

Partial B address = 01024

IR 31
Mode 0

FL = NOT USED

Partial A address = 00496

IR 29
Mode 0

CC for FLOATING POINT SUBTRACT DOUBLE

Assume the index registers contain the following:

IR29 = 0B54 (02900)
IR31 = 0C1C (03100)

After command setup:

Effective A address = 0B54 + 01F0 = 0D44 (03396)
Effective B address = 0C1C + 0400 = 101C (04124)

EXAMPLE 1

Before command execution:

(B) = 41987654321234

(A) = 40987654321234

Step 1:
Invert the sign of (A).

Step 2:
Calculate the difference between the characteristics.
    (a) Difference = 14 or greater
        The larger number = the intermediate sum, bypass steps 3, 4, and 5.
    (b) Difference = less than 14
        The smaller number is right shifted the number of positions indicated
        by the difference. This shifting causes the characteristics to be of
        equal value; zeros are filled into the vacated positions. There is no
        guard digit position; therefore, any character shifted right from the
        least significant position is lost.

Step 3:
Compare the signs of the fractions. If the signs are like, add. If the signs
are unlike, subtract.

Step 4:
Check the intermediate sum for fractional overflow. If overflow, right shift
the intermediate sum one digit position, insert the overflow carry and increase
the characteristic by 1.

Step 5:
Check the characteristic for overflow. If overflow, set the PE indicator.

Step 6:
Check the intermediate sum for all zero sum. If all zero sum, generate a true
zero (64 bits).

Step 7:
If not true zero, normalize the intermediate sum and reduce the characteristic
by the amount of digit positions shifted.

Step 8:
If normalization was done in step 7, check for characteristic underflow. If
underflow, generate a true zero, and set the overflow indicator. Characteristic
underflow does not cause a PE.

Step 9:
Store the result into (B). The sign is governed by the rules of algebra. A
true zero has a positive sign.

Step 10:
Set the appropriate LEG flag to indicate the result's relationship to zero.

	SIGN	CHARACTERISTIC	FRACTION

```
                SIGN ⌐ CHARACTERISTIC                              FRACTION

(B)                0 | 100 0001 | 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000 0001 0010 0011 0100
                     |   4    1 |   9    8    7    6    5    4    3    2    1    0    1    2    3    4

(A)                0 | 100 0000 | 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000 0001 0010 0011 0100
                     |   4    0 |   9    8    7    6    5    4    3    2    1    0    1    2    3    4

Step 1:
(A)                1 | 100 0000 |
                     |   C    0 |

Step 2:
(B)                  | 100 0001 |
(A)                  | 100 0000 |
                     | 000 0001 |

(A)                  |          | 0000 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000 0001 0010 0011
                     |          |   0    9    8    7    6    5    4    3    2    1    0    1    2    3

Step 3:
(B)                0 |          | 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000 0001 0010 0011 0100
(A)                1 |  unlike  | 0000 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000 0001 0010 0011
                     |          | 1000 1110 1110 1110 1110 1110 1110 1110 1110 1111 0001 0001 0001 0001
                     |          |   8    E    E    E    E    E    E    E    E    F    1    1    1    1
```

Step 4:
(Bypass; condition not present)

Step 5:
(Bypass)

Step 6:
Intermediate sum is not all zeros.

Step 7:
Intermediate sum is already normalized.

Step 8:
(Bypass; step 7 did not change characteristic)

```
Step 9:
(B)                0 | 100 0001 | 1001 1000 0111 0110 0101 0100 0011 0010 0001 0000 0001 0010 0011 0100
                     |   4    1 |   9    8    7    6    5    4    3    2    1    0    1    2    3    4

(B)                0 | 100 0001 | 1000 1110 1110 1110 1110 1110 1110 1110 1110 1111 0001 0001 0001 0001
                     |   4    1 |   8    E    E    E    E    E    E    E    E    F    1    1    1    1
```

Step 10:
Set the G flag.

EXAMPLE 2

Before command execution:

(B) = 4211888888888811
(A) = 4190101010101010

Step 1:
    Invert the sign of (A).

Step 2:
    Calculate the difference between the characteristics.
    a) Difference = 14 or greater
       The larger number = the intermediate sum; bypass steps 3,4, and 5.
    b) Difference = less than 14
       The smaller number is right shifted the number of positions indicated
       by the difference. This shifting causes the characteristics to be of
       equal value; zeros are filled into the vacated positions. There is no
       guard digit position; therefore, any character shifted right from the
       least significant position is lost.

Step 3:
    Compare the signs of the fractions. If the signs are like, add. If the
    signs are unlike, subtract.

Step 4:
    Check the intermediate sum for fractional overflow. If overflow, right
    shift the intermediate sum one digit position, insert the overflow carry
    and increase the characteristic by 1.

Step 5:
    Check the characteristic for overflow. If overflow, set the PE indicator.

Step 6:
    Check the intermediate sum for all zero sum. If all zero sum, generate a
    true zero (64 bits).

Step 7:
    If not true zero, normalize the intermediate sum and reduce the character-
    istic by the amount of digit positions shifted.

Step 8:
    If normalization was done in step 7, check for characteristic underflow.
    If underflow, generate a true zero and set the overflow indicator. Char-
    acteristic underflow does not cause a PE.

Step 9:
    Store the result into (B). The sign is governed by the rules of algebra.
    A true zero has a positive sign.

Step 10:
    Set the appropriate leg flag to indicate the result's relationship to
    zero.

```
        CHARACTERISTIC
  SIGN↴     ↓                        FRACTION
      ↓     ↓
(B)   0  100 0010    0001 0001 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 0001 0001
          4    2        1    1    8    8    8    8    8    8    8    8    8    8    1    1

(A)   0  100 0001    1001 0000 0001 0000 0001 0000 0001 0000 0001 0000 0001 0000 0001 0000
          4    1        9    0    1    0    1    0    1    0    1    0    1    0    1    0
Step 1:
(A)   1  100 0001
           C    1


Step 2:
(B)      100 0010
(A)      100 0001
         ───────
         000 0001
(A)                 0000 1001 0000 0001 0000 0001 0000 0001 0000 0001 0000 0001 0000 0001
                       0    9    0    1    0    1    0    1    0    1    0    1    0    1
Step 3:
(B)   0             0001 0001 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 0001 0001
(A)   1  unlike     0000 1001 0000 0001 0000 0001 0000 0001 0000 0001 0000 0001 0000 0001
                    ─────────────────────────────────────────────────────────────────────
                    0000 1000 1000 0111 1000 0111 1000 0111 1000 0111 1000 0111 0001 0000
                       0    8    8    7    8    7    8    7    8    7    8    7    1    0
Step 4:
(Bypass, condition not present)


Step 5:
(Bypass)


Step 6:
Intermediate sum not all zeros.


Step 7:
Intermediate sum   0000 1000 1000 0111 1000 0111 1000 0111 1000 0111 1000 0111 0001 0000
Normalized         1000 1000 0111 1000 0111 1000 0111 1000 0111 1000 0111 0001 0000 0000
         100 0010      8    8    7    8    7    8    7    8    7    8    7    1    0    0
             - 1
         100 0001


Step 8:
Characteristic did not underflow.


Step 9:
(B)   0  100 0010   0001 0001 1000 1000 1000 1000 1000 1000 1000 1000 1000 1000 0001 0001
          4    2       1    1    8    8    8    8    8    8    8    8    8    8    1    1

(B)   0  100 0001   1000 1000 0111 1000 0111 1000 0111 1000 0111 1000 0111 0001 0000 0000
          4    1       8    8    7    8    7    8    7    8    7    8    7    1    0    0
Step 10:
Set the G flag.
```

<u>C = 120(78)(F8)</u>

The contents of the field specified by the effective A address are multiplied by the contents of the field specified by the effective B address. The normalized product is stored in a memory accumulator at locations 328-331. This product (32 bits) in locations 328-331 is then added to the memory accumulator locations 320-323.

With the exceptions noted above, this command functions as the sequential execution of the FLOATING POINT MULTIPLY SINGLE and the FLOATING POINT ADD SINGLE. Refer to the descriptions of these commands for a detailed explanation.

The T portion of the command is not used.

The implied B, after command execution, is equal to 00320. The B portion of the command is not affected.

<u>Command Execution Time</u>

   E = E (F.P. MULTIPLY SINGLE) + E (F.P. ADD SINGLE)

C = 121(79)(F9)

The contents of the field specified by A are multiplied by the contents of the field specified by B. The normalized product is stored in the memory accumulator locations 328-335. This product (64 bits) in locations 328-335 is then added to the memory accumulator locations 320-327.

With the exceptions noted above, this command functions as the sequential execution of the FLOATING POINT MULTIPLY DOUBLE and the FLOATING POINT ADD DOUBLE. Refer to the description of these commands for a detailed explanation.

The T portion of the command is not used.

The implied B after command execution is equal to 00320. The B portion of the command is not affected.

Command Execution Time

   E = E (F.P. MULTIPLY DOUBLE) + E (F.P. ADD DOUBLE)

C = 122(7A)(FA)

General

(A) is compared to (B), and one of the LEG flags is set ON to reflect (A)'s relationship to (B). Comparison is algebraic, taking into account the sign, fraction, and characteristic of each number. The guard digit is used to make the comparison. A characteristic inequality is not decisive for magnitude determination since the fractions may have different numbers of leading zero digits. An equality is established by following the rule for floating point subtraction. Whenever the subtraction would result in, or generate a true zero, the operands are equal. Neither operand is changed as a result of the operation, nor can a characteristic overflow occur.

T is not used; $\underline{B}$ = B

Command Execution Time

E = 2 PO + 12 P + GP

G is the same as G for F.P. ADD SINGLE

C = 123(7B)(FB)

<u>General</u>

(A) is compared to (B), and one of the LEG flags is set ON to reflect (A)'s relationship to (B). Comparison is algebraic, taking into account the sign, fraction, and characteristic of each number. A characteristic inequality is not decisive for magnitude determination since the fractions may have different numbers of leading zero digits. An equality is established by following the rule for floating point subtraction. Whenever the subtraction would result in, or generate a true zero, the operands are equal. Neither operand is changed as a result of the operation, nor can a characteristic overflow occur.

T is not used; $\underline{B} = B$

<u>Command Execution Time</u>

$E = 2\ PO + 20\ P + GP$

G is the same as G for F.P.ADD, SINGLE

C = 124(7C)(FC)

The contents of the field specified by A are multiplied by the contents of the field specified by B. The normalized product is stored in the memory accumulator (locations 320-323).

The multiplication consists of a characteristic addition and a fraction multiplication. The characteristic of the product is derived by adding the A operand characteristic to the B operand characteristic and subtracting a hexadecimal 40.

If the signs of the operands are alike, the sign of the product is positive; if the signs of the operands are unlike, the sign of the product is negative.

If either operand contains an all zero fraction, the product will consist of a true zero (32 zero bits).

Prior to performing the multiplication, the operands are normalized if necessary and the relative characteristic is adjusted accordingly. These operands are not disturbed in their original memory locations if normalization is performed.

The operands are multiplied resulting in an intermediate product. The intermediate product is tested to see whether normalization is required before storing the final product in the memory accumulator. The final product fraction consists of six hexadecimal digits.

If the final product characteristic exceeds hexadecimal 7F, a PE trap is taken and memory accumulator locations 320-323 are in an unpredictable state. The overflow trap does not occur for an intermediate product characteristic exceeding hexadecimal 7F if the final characteristic can be brought within range by normalization.

If the intermediate or final product characteristic is less than hexadecimal 00, underflow occurs and a true zero is generated. The true zero is stored in the memory accumulator as the final product, the overflow indicator (OF) is set ON and the command terminates.

The T portion of the command is not used.

The implied B after command execution is equal to 00320. The B portion of the command is not affected.

Command Execution Time

$E = 18PO + 68P + [6AP + 12BP + 6CP + 12DP]$

A = Number of A Field Digits = 1, 2, 4, 8, C, E, F with neighboring right hand digit $\leq$ Hex A.

B = Number of A field digits = 3, 5, 6, 7, 9, A, B, D with neighboring right hand digit $\leq$ Hex A.

C = Number of A field digits = 0, 1, 3, 7, B, D, E with neighboring right hand digit > Hex A.

D = Number of A field digits = 2, 4, 5, 6, 8, 9, A, C with neighboring right digits $\leq$ Hex A.

Note the sum of A + B + C + D must be $\leq$ 6. It is assumed that normalization is not necessary.

The equation for the average execution time of F. P. MULTIPLY SINGLE is:

$$E = 18\ PO + 120P + X\ [PO + 3P + \frac{1+N}{2}P] + Y\ [PO + 4P + \frac{M}{2}P]$$

$$+ T\ [PO + 3P + \frac{1+N}{2}P] + R\ [PO + 4P + \frac{M}{2}P] - Z\ [2PO + 14P]$$

HEXIDECIMAL MULTIPLICATION TABLE

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

```
 Q     RA     A2     A1     T     RB     B2     B1
 7C     78     00     C8     00     78     00     CC
```

```
                                            |
                                            Partial B address = 00204

                                     |
                                     IR 30
                                     Mode 0

                          |
                          FL = NOT USED

                 |
                 Partial A address = 00200

        |
        IR 30
        Mode 0

 |
 CC for 2-address FLOATING POINT MULTIPLY SINGLE
```
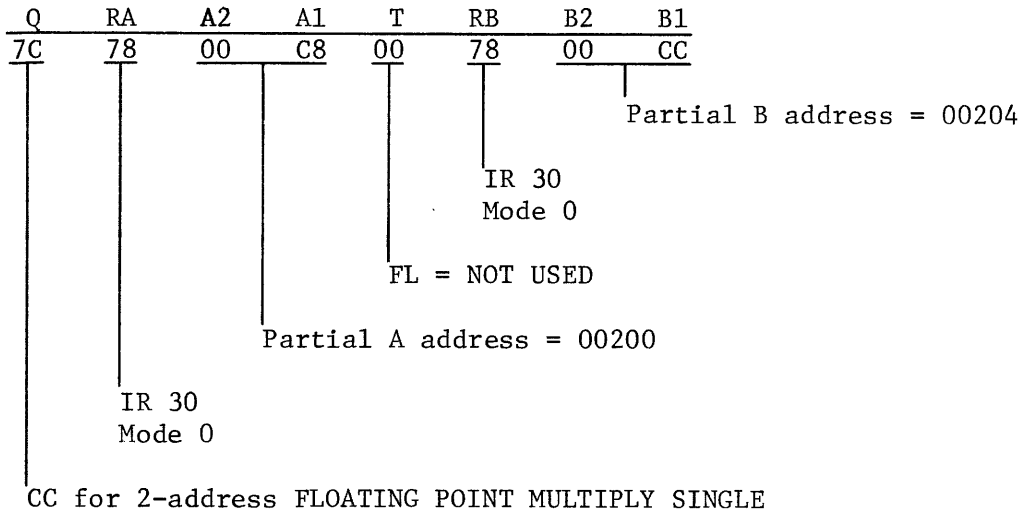
Assume the index registers contain the following:

(IR30) = 1388 (05000)

After command setup:

Effective A address = 1388 + 00C8 = 1450 (05200)
Effective B address = 1388 + 00CC = 1454 (05204)

The A and B operands remain unchanged.

A	1450	1451	1452	1453
(A)	44	CA	FE	00

B	1454	1455	1456	1457
(B)	42	3D	00	00

EXAMPLE 1   (Like signs, normalized whole numbers)

Add A operand characteristic to B operand characteristic.

$$\begin{array}{r} 44 \\ + \ 42 \\ \hline 86 \end{array}$$

Subtract hexadecimal 40 from result.

Intermediate product characteristic.

$$\begin{array}{r} 86 \\ - \ 40 \\ \hline 46 \end{array}$$

Multiply.

$$\begin{array}{r} CAFE \\ 3D \\ \hline A4EE6 \\ 260FA \\ \hline \end{array}$$

Intermediate product.

305E86

Store characteristic and final product
in the memory accumulator

0140	0141	0142	0143
46	30	5E	86

EXAMPLE 2   (Unlike signs, non-normalized fractional numbers)

Assume command format and setup are the same as in Example 1, but that the A and B operands are as follows:

A
(A)

1450	1451	1452	1453
42	00	50	C0

The A and B operands remain
unchanged

B
(B)

1454	1455	1456	1457
C3	00	00	50

Add A operand characteristic to B operand characteristic.

```
      42
    + 43
      85
```

Subtract hexadecimal 40 from result.

```
      85
    - 40
```

Intermediate characteristic before normalization

```
      45
```

Determine the sign of the product. (pos. x neg. = neg.)   80

Count leading zeros of both fractions.   6

Subtract number of leading zeros from intermediate characteristic.   45

```
     -  6
```

Adjusted intermediate characteristic

```
       39
```

Add sign.

```
       39
     + 80
       B9
```

Normalize fractions internally.

```
     50  C0  00
     50  00  00
```

Multiply.

```
       50C
         5
```

Final product

```
      193C
```

Store characteristic and final product
in the memory accumulator.

0140	0141	0142	0143
B9	19	3C	00

C = 125(7D)(FD)

The contents of the field specified by A are multiplied by the contents of the
field specified by B. The normalized product is stored in the memory accumula-
tor (locations 320-327).

The multiplication consists of a characteristic addition and a fraction multi-
plication. The characteristic of the product is derived by adding the A oper-
and characteristic to the B operand characteristic and subtracting a hexadeci-
mal 40.

If the signs of the operands are alike the sign of the product is positive; if
the signs of the operands are unlike the sign of the product is negative.

If either operand contains an all zero fraction, the product will consist of
a true zero (64 zero bits).

Before performing the multiplication, the operands are normalized if necessary
and the relative characteristic is adjusted accordingly. These operands are
not disturbed in their original memory locations if normalization is performed.

The operands are multiplied resulting in an intermediate product. The inter-
mediate product is tested to see whether normalization is required before stor-
ing the final product in the memory accumulator. The final product fraction
consists of 14 hexadecimal digits.

If the final product characteristic exceeds hexadecimal 7F, a PE trap is taken
and memory accumulator locations 320-327 are in an unpredictable state. The
overflow trap does not occur for an intermediate product characteristic exceed-
ing hexadecimal 7F if the final characteristic can be brought within range by
normalization.

If the intermediate or final product characteristic is less than hexadecimal
00, underflow occurs and a true zero is generated. The true zero is stored in
the memory accumulator as the final product, the overflow indicator (OF) is
set ON, and the command terminates.

The T portion of the command is not used.

The implied B after command execution is equal to 00320. The B portion of the
command is not affected.

Command Execution Time

$$E = 38PO + 229P + [12AP + 24BP + 12CP + 24DP]$$

A, B, C, D are the same as F.P. MULTIPLY SINGLE and there is no normalization.

The equation for the <u>average</u> execution time of F.P. MULTIPLY DOUBLE is:

$$E = 38PO + 471P + X[PO + 3P + \frac{1 + N}{2}P] + Y[PO + 4P + \frac{M}{2}P]$$

$$+ T[PO + 3P + \frac{1 + N}{2}P] + R[PO + 4P + \frac{M}{2}P] - Z[2PO + 29P]$$

X = 1; If prenormalization is required on the B field with an odd number of leading zeros.

Y = 1; If prenormalization is required on the B field with an even number of leading zeros.

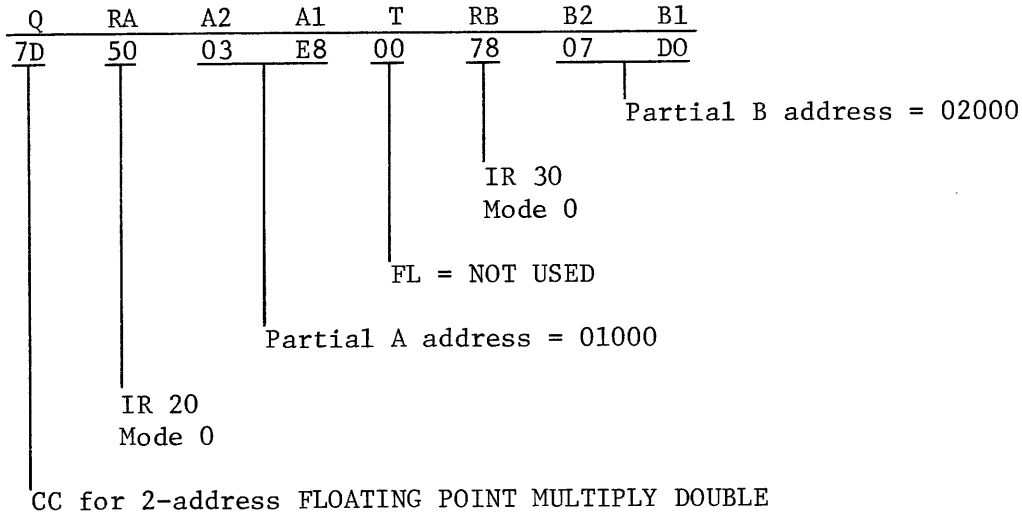T = 1; If prenormalization is required on the A field with an odd number of leading zeros.

R = 1; If prenormalization is required on the A field with an even number of leading zeros.

Z = Value dependent on the number of leading zeros in the A field, i.e., $0 < Z < 7$ for single precision and $0 < Z < 14$ for double precision.

M = Number of even leading zeros.

N = Number of odd leading zeros.

X and Y are mutually exclusive.

```
  Q     RA    A2    A1    T    RB    B2    B1
 7D     50    03    E8    00   78    07    DO
```

Partial B address = 02000

IR 30
Mode 0

FL = NOT USED

Partial A address = 01000

IR 20
Mode 0

CC for 2-address FLOATING POINT MULTIPLY DOUBLE

Assume the index registers contain the following:

(IR 20) = 2710 (10000)
(IR 30) = 3A98 (15000)

After command setup:

Effective A address = 2710 + 03E8 = 2AF8 (11000)
Effective B address = 3A98 + 07DO = 4268 (17000)

The A and B operands remain unchanged.

A	2AF8	2AF9	2AFA	2AFB	2AFC	2AFD	2AFE	2AFF
(A)	4E	94	C3	5D	2A	B7	F8	6E

B	4268	4269	426A	426B	426C	426D	426E	426F
(B)	4E	00	00	00	00	EF	33	19

Add A operand characteristic to B operand characteristic   4E
                                                           + 4E
                                                             ‾‾
                                                             9C

Subtract hexadecimal 40 from result.   9C
                                                           - 40
                                                             ‾‾
Intermediate characteristic before normalization   5C

Determine the sign of the product. (pos. x pos. = pos.)   00

Count leading zeros of both fractions.   8

Subtract number of leading zeros from intermediate characteristic. 5C
                                                           - 8
                                                             ‾‾
Adjusted intermediate characteristic   54

Normalize fractions internally.   94 C3 5D 2A B7 F8 6E
                                     EF 33 19 00 00 00 00

Multiply   94C35D2AB7F86E
                                                   EF3319
                                          ‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾
                                          53ADE468077BBDE
                                          94C35D2AB7F86E
                                          1BE4A178027E94A
                                          1BE4A178027E94A
                                          8B7727580C78E72
                                          822AF18560F9604
                                          ‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾‾
Intermediate product   8B00156D855CC1272CBE

                    Truncated ──────────────────────────────┘

Final product   8B00156D855CC1

Store characteristic and final product
in the memory accumulator.

0140	0141	0142	0143	0144	0145	0146	0147
54	8B	00	15	6D	85	5C	C1

C = 126(7E)(FE)

The contents of the fields specified by A and B are considered to be 4-byte hexadecimal fields in single-precision floating-point format. These operands may or may not be normalized; however, a saving in command execution time is gained if the operands are normalized. T is not used in this command.

The contents of the field specified by B (the dividend) are divided by the contents of field specified by A (the divisor). A normalized result (the quotient) is stored in memory locations 320 through 323. No remainder is preserved. The sign of the quotient is determined by the rules of algebra, i.e., like signs result in a positive quotient, unlike signs result in a negative quotient.

The division operation consists of normalizing the operands (if they are not normalized), subtracting the characteristics, determining the sign, dividing the fractions, and storing the normalized quotient.

The dividend characteristic less the divisor characteristic, plus a binary 64, is used as an intermediate quotient characteristic. If the division of the fractions results in an intermediate quotient fraction that has leading zeros, the intermediate quotient fraction will be normalized and the intermediate quotient characteristic will be adjusted accordingly. If the division of the dividend and divisor fractions resulted in an intermediate quotient fraction with no leading zeros, the intermediate quotient characteristic and fraction require no adjustment and, therefore, are stored as the final quotient characteristic and fraction.

All dividend fraction digits participate in forming the quotient fraction, even if the normalized dividend fraction is larger than the normalized divisor fraction. The quotient fraction is truncated to 6 hexadecimal digits and no remainder is preserved.

After command execution the command and operands are unchanged. The implied B (LB address register) is equal to 00320 (decimal) for purposes of chaining commands.

NOTE

> During execution of F. P. Divide single the contents of memory addresses 328-335 may be disturbed.

ABNORMAL CONDITIONS

● Memory addresses of a floating point operand (dividend or divisor) not zero modulo 4

  In this case, the command is aborted and a PE trap occurs. The accumulator and instruction are undisturbed.

● Division by zero is attempted

  If the divisor fraction is equal to zero, the command is aborted and a PE trap occurs. The accumulator and B portion of the instructions are undisturbed.

- **The final quotient characteristic exceeds binary 127 (larger than +63)**

  A PE trap occurs and unpredictable data is stored in the quotient memory area.

- **The final quotient characteristic is approaching zero (less than -64)**

  Overflow indicator (OF) is turned ON, a true zero is generated and placed in the quotient memory area (true zero equals 32 zero bits).

- **Division into zero is attempted**

  Overflow indicator is not turned ON, but a true zero is stored in the quotient memory area.

Command Execution Time
$$E = E_1 + E_2$$

Where $E_1 = 16 \text{ PO} + 209P$

E_2 = Additional time required if dividend and/or divisor are not normalized (operands do not have to be normalized since pre-normalizing occurs during the command if required).

$$E_2 = 18PO + 120P + X \left(PO + 3P + \frac{1 + N}{2}P\right) + Y \left(PO + 4P + \frac{M}{2}P\right)$$
$$+ T \left(PO + 3P + \frac{1 + N}{2}P\right) + R \left(PO + 4P + \frac{M}{2}P\right) - Z \left(2PO + 14P\right)$$

$X = 1$, if prenormalization is required on the B field with an odd number of leading zeros.

$Y = 1$, if prenormalization is required on the B field with an even number of leading zeros.

$T = 1$, if prenormalization is required on the A field with an odd number of leading zeros.

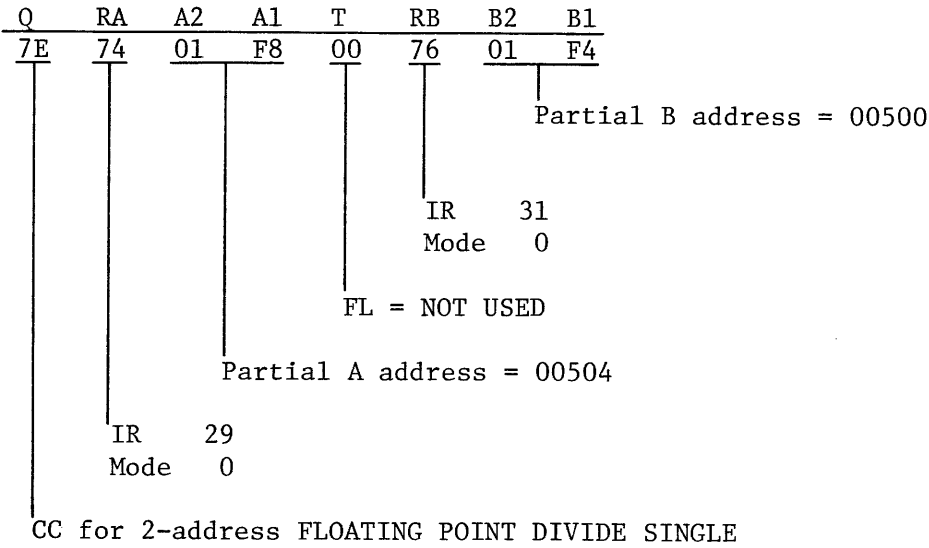$R = 1$, if prenormalization is required on the A field with an even number of leading zeros.

$Z =$ Value dependent on the number of leading zeros in the A field, i.e., $0 < Z < 7$ for single precision and $0 < Z < 14$ for double precision.

$M =$ Number of even leading zeros.

$N =$ Number of odd leading zeros.

X and Y are mutually exclusive.

T and R are mutually exclusive.

```
 Q    RA   A2   A1    T    RB   B2   B1
 7E   74   01   F8    00   76   01   F4
```

Partial B address = 00500

IR   31
Mode   0

FL = NOT USED

Partial A address = 00504

IR   29
Mode   0

CC for 2-address FLOATING POINT DIVIDE SINGLE

Assume the index registers contain the following:

        IR29 = 0B54 = (02900 in decimal)
        IR31 = 0C1C = (03100 in decimal)

After command setup:

Effective A address = 0B54 + 01F8 = 0D5C = (03420 decimal)
Effective B address = 0C1C + 01F4 = 0E10 = (03600 decimal)

Before command execution:

A =	0D5C	0D5D	0D5E	0D5F
(A) =	45	00	A0	00

B =	0E10	0E11	0E12	0E13
(B) =	48	06	40	00

The implied B (LB address register) = 00320 for purposes of chaining after command execution.

EXAMPLE 1 (Like signs, unnormalized whole numbers)

Subtract A-operand characteristic from B-operand characteristic (not including sign bit).

$$\begin{array}{r} 48 \\ -45 \\ \hline 03 \end{array}$$

Add hexadecimal 40 to result (excess binary 64).

Intermediate quotient characteristic

$$\begin{array}{r} 03 \\ +40 \\ \hline 43 \end{array}$$

Normalize the divisor fraction if leading zeros are present.

$$\begin{array}{r} 00A000 \\ = A00000 \end{array}$$

Add the number of leading zeros that were present in the divisor fraction to the intermediate quotient characteristic.

$$\begin{array}{r} 43 \\ +02 \\ \hline 45 \end{array}$$

Normalize the dividend fraction if leading zeros are present.

$$\begin{array}{r} 064000 \\ = 640000 \end{array}$$

Subtract the number of leading zeros that were present in the dividend fraction from the intermediate quotient characteristic. New intermediate quotient characteristic

$$\begin{array}{r} 45 \\ -01 \\ \hline 44 \end{array}$$

Divide the B operand fraction by the A operand fraction.

```
        0A0000
   A)640000
       64
      ------
      000000
```

Intermediate quotient fraction

0A0000

Normalize the intermediate quotient fraction if leading zeros are present, truncate at 6th hexadecimal position.

A00000

Final quotient fraction

A00000

Subtract the number of leading zeros that were present in the quotient fraction from the present intermediate quotient characteristic.

$$\begin{array}{r} 44 \\ -01 \\ \hline 43 \end{array}$$

Final quotient characteristic

43

Determine the state of the quotient sign bit by comparing the sign bits of the A and B operands.

0

Store the sign, final quotient characteristic, and final quotient fraction in memory locations 00320 through 00323 (decimal), hexadecimal addresses 0140 through 0143

0140	0141	0142	0143
43	A0	00	00

The contents of A and B are unchanged.

EXAMPLE 2 (Unlike signs, one unnormalized operand, truncation)

Assume the operand fields in the preceding example contained the following information:

A	0D5C	0D5D	0D5E	0D5F
(A)	53	00	00	18
B	0E10	0E11	0E12	0E13
(B)	C9	27	10	00

(note that the B operand is a negative quantity)

Subtract A operand characteristic from B operand characteristic (excluding sign bit, C9 becomes 49 with sign bit removed).

      53
     -49

      04

Add hexadecimal 40 to result to get the initial intermediate quotient characteristic.
Intermediate quotient characteristic

      04
     +40

      44

Normalize divisor fraction if leading zeros are present.

   000018
   180000

Add the number of leading zeros that were present in the divisor fraction to the intermediate quotient characteristic.
New intermediate quotient characteristic

      44
     +04

      48

Normalize dividend fraction if leading zeros are present (not required in this example since fraction is normalized).

   271000
   271000

Subtract the number of leading zeros that were present in the dividend fraction from the intermediate quotient characteristic (not required here). Intermediate quotient characteristic remains the same.

      48
     -00

      48

Divide B-operand fraction by A-operand fraction. This results in an unending fraction and, since this is single precision, only 6 hexadecimal places are computed. Digits beyond the sixth position are considered to be insignificant.

```
           01A0AA
       18)271000
           18
          -----
           0F1
            F0
          -----
           0100
             F0
           -----
            100
             F0
```

Intermediate quotient fraction

   01A0AA

Normalize the intermediate quotient fraction if leading zeros are present; truncate at 6th hexadecimal position, and zero fill to right.

   01A0AA
   1A0AA0

Final quotient fraction

   1A0AA0

Subtract the number of leading zeros that were in the inter-   48
mediate quotient fraction from the present intermediate quotient   −01
characteristic.   47

Compare sign bits of operands A and B to determine the sign bit   1
of the quotient (unlike operand sign bits result in a negative
quotient sign (bit = 1) stored in the eighth bit position of the
quotient characteristic).

Store sign bit, final quotient characteristic, and final quotient
fraction, in memory locations 00320 through 00323 (decimal),
hexadecimal addresses 0140 through 0143.

0140	0141	0142	0143
07	1A	CA	A0

NOTES

Normalized Operands: Execution time is saved since
prenormalization is not required during the command.

Unnormalized Operands: Additional execution is required
since prenormalization is necessary during command
operation.

Quotients: Quotients are stored in the memory accumulator
at memory location 00320 - 00323, in normalized format.

Fractional Numbers: Fractional numbers are manipulated
in exactly the same manner as floating point representa-
tions of whole numbers.

C = 127(7F)(FF)

Floating point divide with double precision functions in the same manner as floating point divide with single precision. The only difference between the two instructions is precision. Operands and quotients for floating point divide double are 8-byte hexadecimal expressions (as opposed to 4 bytes in single precision). The quotient is stored in the memory accumulator at locations 00320 through 00327 (as opposed to 00320 through 00323 as in single precision).

Command Execution Time:
$$E = E_1 + E_2$$

Where $E1 = 32 \ PO + 893P$

$E2$ = Additional time required if dividend and/or divisor are not normalized, and normalization takes place during the command.

$$E2 = 38PO + 471P + X \left[PO + 3P + \frac{1 + N}{2} P\right] + Y \left[PO + 4P + \frac{M}{2} P\right]$$

$$+ T \left[PO + 3P + \frac{1 + N}{2} P\right] + R \left[PO + 4P + \frac{M}{2} P\right] - Z (2PO + 29P)$$

$X = 1$; If prenormalization is required on the B field with an odd number of leading zeros.

$Y = 1$; If prenormalization is required on the B field with an even number of leading zeros.

$T = 1$; If prenormalization is required on the A field with an odd number of leading zeros.

$R = 1$; If prenormalization is required on the A field with an even number of leading zeros.

$Z =$ Value dependent on the number of leading zeros in the A field, i.e., $0 < Z < 7$ for single precision and $0 < Z < 14$ for double precision.

$M =$ Number of even leading zeros.

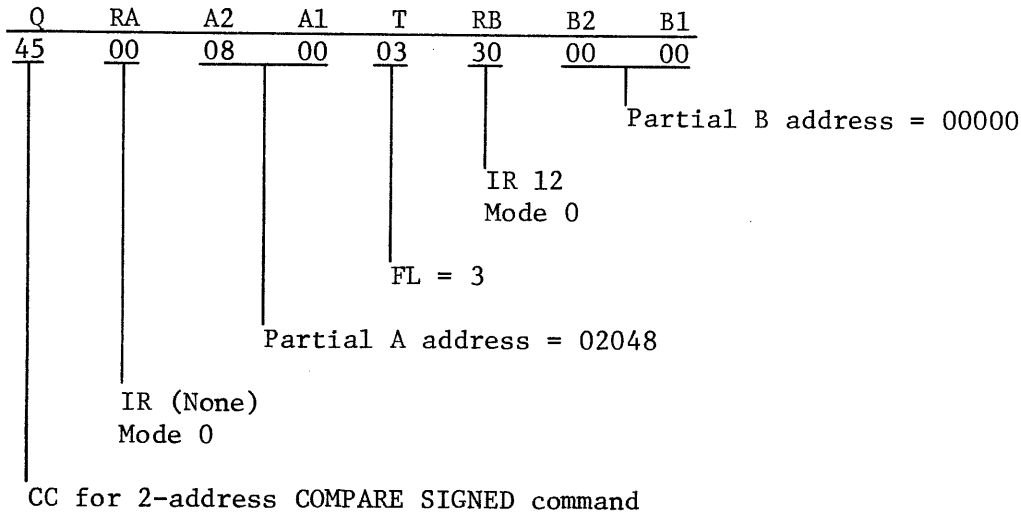$N =$ Number of odd leading zeros.

X and Y are mutually exclusive.

T and R are mutually exclusive.

The introduction to this publication contains formulas for determining total hardware command time. The following examples are offered as an aid to using the formulas.

EXAMPLE 1

Q	RA	A2	A1	T	RB	B2	B1
45	00	08	00	03	30	00	00

Partial B address = 00000

IR 12
Mode 0

FL = 3

Partial A address = 02048

IR (None)
Mode 0

CC for 2-address COMPARE SIGNED command

According to the basic formula:

C (Total command time) = S (setup time) + E (execution time)

The following formula is used to find the setup time:

$$S = (M_1A + 1)(PO + 2P) + RAP + M_2A(PO + 3P) + 2M_3A +$$
$$K[(M_1B + 1)(PO + 2P) + RBP + M_2B(PO + 3P) + 2M_3BP]$$

The variables in this setup time formula assume the following values for the given command: (NSEC. = NANOSECONDS)

$M_1A = 0$ (Not mode 1 addressing)

PO = 290 nsec. (Minimum adder propagation time assumed for this example)

P = 760 nsec.

$M_2A = 0$ (Not mode 2 addressing)

RA = 0 (No indexing required, no index register specified)

$M_3A = 0$ (Not mode 3 indexing)

K = 1 (2-address format)

$M_1B = 0$ (Not mode 1 addressing)

$M_2B = 0$ (Not mode 2 addressing)

$RB = 1$ (Indexing required once, mode 0 addressing and an index register is specified)

$M_3B = 0$ (Not mode 3 indexing)

Placing these values in the formula gives:

S = 1 (290 nsec. + 1520 nsec.) + 1 [1 (290 nsec. + 1520 nsec.) + 1 ˙ 760 nsec.]

S = 1810 nsec. + 1 [1810 nsec. + 760 nsec.]

S = 1810 nsec. + 2570 nsec.

S = 4380 nsec. = 4.38 μs.

The following formula is used to find the execution time of a COMPARE SIGNED command:

E = 2PO + 4P + 3TP

E = 580 nsec. + 3040 nsec. + 6840 nsec.

E = 10460 nsec. = 10.46 μs.

Since the setup and execution time of the given command are now known, the total command time can be obtained as follows:

C = S + E

C = 4380 nsec. + 10460 nsec.

C = 14840 nsec. = 14.84 μs.

The total command time in the foregoing example was calculated assuming the minimum adder propagation time (value assigned to PO in the setup and execution formulas). To get a true picture of the range within which the total command time may vary, the same command will be recalculated assuming the maximum adder propagation time (PO = 325 nsec.).

Assuming the maximum value for PO (325 nsec.), the setup time becomes:

$S = (M_1A + 1)(PO + 2P) + RAP + M_2A(PO + 3P) + 2M_3A +$
$\quad K[(M_1B + 1)(PO + 2P) + RBP + M_2B(PO + 3P) + 2M_3BP]$

S = 1 (325 nsec. + 1520 nsec.) + 1 [1 (325 nsec. + 1520 nsec.) + 1 ˙ 760 nsec.]

S = 1845 nsec. + 1 [1845 nsec. + 760 nsec.]

S = 1845 nsec. + 2605 nsec.

S = 4450 nsec. = 4.45 μs.

The execution time becomes:

$$E = 2PO + 4P + 3TP$$

$$E = 2 \cdot 325 \text{ nsec.} + 4 \cdot 760 \text{ nsec.} + 3 \cdot 3 \cdot 760 \text{ nsec.}$$

$$E = 650 \text{ nsec.} + 3040 \text{ nsec.} + 6840 \text{ nsec.}$$

$$E = 10530 \text{ nsec.} = 10.53 \text{ μs.}$$

The resulting total command time for the given command is:

$$C = S + E$$

$$C = 4450 \text{ nsec.} + 10530 \text{ nsec.}$$

$$C = 14980 \text{ nsec.} = 14.98 \text{ μs.}$$

The total command time (14.98 μs.) was calculated assuming the maximum adder propagation time. This total command time should now be compared with the previously calculated total command time which assumed the minimum adder propagation time. This results in a range within which the total command time may vary. For the given command, the total command time may vary from 14.84 μs. to 14.98 μs., depending upon the actual adder propagation time that occurs.

EXAMPLE 2

Q	RA	A2	A1	T	RB	B2	B1
4C	66	01	F4	07	40	00	C8

Partial B address = (00200)

IR 16
Mode 0

FL = 7

Partial A address = (00500)

IR 25
Mode 2

CC for 2-address PACK command

According to the basic formula:

C (Total command time) = S (Setup time) + E (Execution time)

The following formula is used to find the setup time:

$$S = (M_1A + 1)(PO + 2P) + {_{RAP}} + M_2A(PO + 3P) + 2M_3A +$$
$$K[(M_1B + 1)(PO + 2P) + RBP + M_2B(PO + 3P) + 2M_3BP]$$

The variables in this setup time formula assume the following values for the given command:

$M_1A = 0$ (Not mode 1 addressing)

$PO = 290$ nsec. (Minimum adder propagation time assumed for this example)

$P = 760$ nsec.

$M_2A = 1$ (Mode 2 addressing used for A operand)

$RA = 1$ (Indexing required once, mode 2 addressing and an index register are specified)

$M_3A = 0$ (Not mode 3 indexing)

$K = 1$ (2-address format)

$M_1B = 0$ (Not mode 1 addressing)

$M_2B = 0$ (Not mode 2 addressing)

$RB = 1$ (Indexing required once, mode 0 addressing and an index register are specified)

$M_3B = 0$ (Not mode 3 indexing)

Placing these values in the formula gives:

$S = 1$ (290 nsec. + 1520 nsec.) + 1 $\cdot$ 760 nsec. + 1 (290 nsec. + 2280 nsec.)
+ 1 [1 (290 nsec. + 1520 nsec.) + 1 $\cdot$ 760 nsec.]

$S = 1810$ nsec. + 760 nsec. + 2570 nsec. + 1 [1810 nsec. + 760 nsec.]

$S = 1810$ nsec. + 760 nsec. + 2570 nsec. + 2570 nsec.

$S = 7710$ nsec. $= 7.71$ µs.

The following formula is used to find the execution time of a PACK command with a T value that is odd.

$$E = PO + \left[\frac{3\ (T + 1)}{2} \right]\ P$$

$$E = 290 \text{ nsec.} + \left[\frac{3\ (7 + 1)}{2} \right]\ 760 \text{ nsec.}$$

$$E = 290 \text{ nsec.} + \left[\frac{3 \cdot 8}{2} \right]\ 760 \text{ nsec.}$$

$$E = 290 \text{ nsec.} + \left[\frac{24}{2} \right]\ 760 \text{ nsec.}$$

$E = 290$ nsec. + 12 $\cdot$ 760 nsec.

$E = 290$ nsec. + 9120 nsec.

$E = 9410$ nsec. $= 9.41$ µs.

Since the setup and execution time of the given command are now known, the total command time can be obtained as follows:

$$C = S + E$$

$$C = 7710 \text{ nsec.} + 9410 \text{ nsec.}$$

$$C = 17120 \text{ nsec.} = 17.12 \text{ } \mu s.$$

The total command time was calculated assuming the minimum adder propagation time (value assigned to PO in the setup and execution formulas). To get a true picture of the range within which the total command time may vary, the same command will be recalculated assuming the maximum adder propagation time (PO = 325 nsec.).

Assuming the maximum value for PO (325 nsec.), the setup time would then become:

$$S = (M_1A + 1)(PO + 2P) + RAP + M_2A(PO + 3P) + 2M_3A +$$

$$K[(M_1B + 1)(PO + 2P) + RBP + M_2B(PO + 3P) + 2M_3BP]$$

$$S = 1 (325 \text{ nsec.} + 1520 \text{ nsec.}) + 1 \cdot 760 \text{ nsec.} + 1 (325 \text{ nsec.} + 2280 \text{ nsec.})$$

$$+ 1 [1 (325 \text{ nsec.} + 1520 \text{ nsec.}) + 1 \cdot 760 \text{ nsec.}]$$

$$S = 1845 \text{ nsec.} + 760 \text{ nsec.} + 2605 \text{ nsec.} + 1 [1845 \text{ nsec.} + 760 \text{ nsec.}]$$

$$S = 1845 \text{ nsec.} + 760 \text{ nsec.} + 2605 \text{ nsec.} + 2605 \text{ nsec.}$$

$$S = 7815 \text{ nsec.} = 7.815 \text{ } \mu s.$$

The execution time would become:

$$E = PO + \left[\frac{3(T + 1)}{2} \right] P$$

$$E = 325 \text{ nsec.} + \left[\frac{3(7 + 1)}{2} \right] 760 \text{ nsec.}$$

$$E = 325 \text{ nsec.} + \left[\frac{3 \cdot 8}{2} \right] 760 \text{ nsec.}$$

$$E = 325 \text{ nsec.} + \left[\frac{24}{2} \right] 760 \text{ nsec.}$$

$$E = 325 \text{ nsec.} + 12 \cdot 760 \text{ nsec.}$$

$$E = 325 \text{ nsec} + 9120 \text{ nsec.}$$

$$E = 9445 \text{ nsec.} = 9.445 \text{ } \mu s.$$

The resulting total command time for the given command would be as follows:

C = S + E

C = 7815 nsec. + 9445 nsec.

C = 17260 nsec. = 17.26 μs.

The total command time (17.26 μs.) was calculated assuming the maximum adder propagation time. This total command time should now be compared with the previously calculated total command time which assumed the minimum adder propagation time. This results in a range within which the total command time may vary. For the given command, the total command time may vary from 17.12 μs. to 17.26 μs., depending upon the actual adder propagation time that occurs.

NCR REFERENCE MANUAL

An Educational Publication

NCR CENTURY 201 PROCESSOR

This publication contains the functional differences, involving the trunk con-
figuration, trunk transfer rates, and system I/O bandwidth, between the NCR
Century 200 processor and the NCR Century 201 processor. The NCR Century 200
processor product information publication should be used in conjunction with
this publication for a complete description of processor functions.

GENERAL DESCRIPTION

The NCR Century 201 contains interfacing buffers between one or, as an option,
two I/O trunks and memory. The interface comprises a 32-character input/output
buffer for either trunk 4 or both trunks 3 and 4. Data transfer between memo-
ry and the buffers takes place two bytes at a time, which effectively doubles
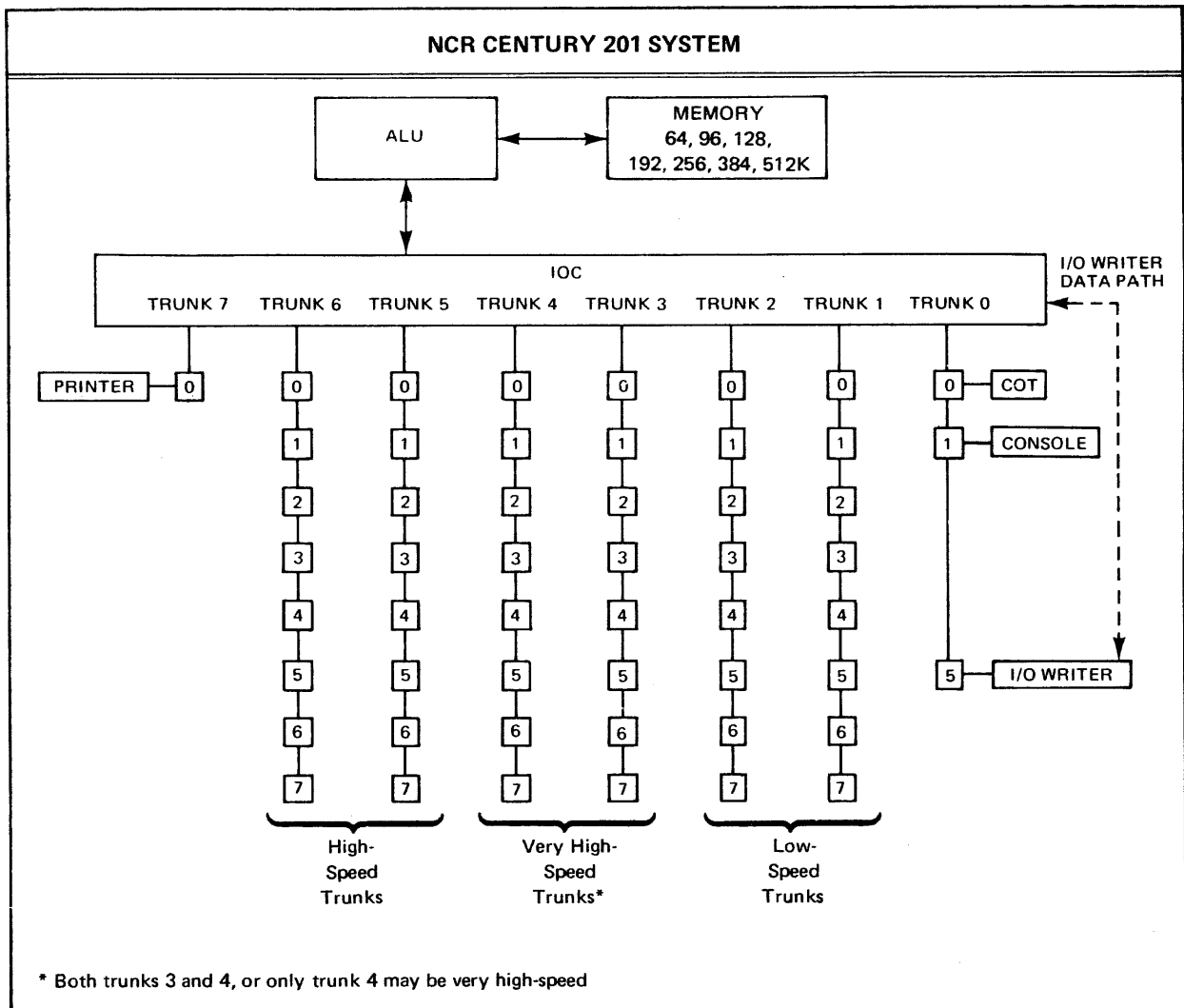the existing trunk transfer rate and system I/O bandwidth of the NCR Century
200.

The increased I/O capabilities of the NCR Century 201 processor allow the use
of high-density NCR 657 disc units for faster data throughput and increased
storage capacity. By being able to accommodate many high-speed magnetic file
devices, the NCR Century 201 provides a powerful processing system for multi-
programming applications. All features available for the NCR Century 200 are
available for the NCR Century 201, such as extended memory, floating point, and
others.

Trunk Configuration

The NCR Century 201 is an octaplex system with two high-speed trunks and one
very high-speed trunk (trunk 4). Optionally, trunk 3 may also be converted to
a very high-speed trunk.

Either trunk 4 or trunks 3 and 4, modified with the buffer interface, are
designated as very high-speed (VHS) trunks; trunks 5 and 6 are the high-speed
(HS) trunks; trunks 0 and 7 are dedicated to integrated peripherals; trunks 1
and 2 remain the low-speed trunks of the system. (If only trunk 4 is modified
to become the very high-speed trunk, trunk 3 remains a low-speed trunk.)

The following illustration shows the system configuration of the NCR Century
201.

NCR CENTURY 201 SYSTEM

```
                    ┌──────────┐              ┌──────────────────┐
                    │          │              │     MEMORY       │
                    │   ALU    │ ◄──────────► │  64, 96, 128,    │
                    │          │              │ 192, 256, 384, 512K │
                    └──────────┘              └──────────────────┘
                         ▲
                         │
                         ▼
    ┌─────────────────────────────────────────────────────────────────────┐      I/O WRITER
    │                              IOC                                      │      DATA PATH
    │  TRUNK 7  TRUNK 6  TRUNK 5  TRUNK 4  TRUNK 3  TRUNK 2  TRUNK 1  TRUNK 0 │◄─ ─ ┐
    └─────────────────────────────────────────────────────────────────────┘      │
```

PRINTER –│0│	│0│	│0│	│0│	│G│	│0│	│0│	│0│–COT	
	│1│	│1│	│1│	│1│	│1│	│1│	│1│–CONSOLE	
	│2│	│2│	│2│	│2│	│2│	│2│		
	│3│	│3│	│3│	│3│	│3│	│3│		
	│4│	│4│	│4│	│4│	│4│	│4│		
	│5│	│5│	│5│	│5│	│5│	│5│	│5│–I/O WRITER	
	│6│	│6│	│6│	│6│	│6│	│6│		
	│7│	│7│	│7│	│7│	│7│	│7│		

```
              ⎨_____⎬   ⎨_____⎬   ⎨_____⎬
                  High-          Very High-        Low-
                  Speed            Speed           Speed
                  Trunks          Trunks*          Trunks
```

* Both trunks 3 and 4, or only trunk 4 may be very high-speed

Trunk Transfer Rates

Installing the interface to trunks 3 and 4 doubles the maximum trunk transfer rate and the system I/O bandwidth of the NCR Century 200.

The following table lists the trunk bandwidths of the NCR Century 201.

NCR CENTURY 201 TRUNK BANDWIDTHS	
Trunk	**Bandwidth**
Low-speed	120 KB
High-Speed	487 KB
Very high-speed	900 KB

System I/O Bandwidth

The maximum I/O transfer rate of the NCR Century 201 is: 1387 KB with 1 VHS trunk, 1700 KB with 2 VHS trunks.

The following three example I/O configurations help illustrate the expanded I/O capabilities of the NCR Century 201.

EXAMPLE 1:

    Trunk 7 - 640-200 Integrated Printer - 77 KB
    Trunk 6 - 655-201 Dual Disc Unit - 108 KB
    Trunk 5 - 633-211 Tape Handler - 144 KB
    Trunk 4 - 657-102 High-Density Disc Unit - 500 KB

EXAMPLE 2:

    Trunk 7 - 640-200 Integrated Printer - 77 KB
    Trunk 6 - 684-101 Card Reader/Punch - 50 KB
    Trunk 5 - 633-119 Tape Handler - 40 KB
    Trunk 4 - 657-102 High-Density Disc Unit - 500 KB
    Trunk 3 - 657-102 High-Density Disc Unit - 500 KB

EXAMPLE 3:

    Trunk 6 - 633-311 Tape Handler - 240 KB
    Trunk 5 - 633-211 Tape Handler - 144 KB
    Trunk 4 - 657-102 High-Density Disc Unit - 500 KB
    Trunk 3 - 657-102 High-Density Disc Unit - 500 KB
    Trunk 2 - Freestanding Printer - 38 KB
    Trunk 1 - Freestanding Printer - 38 KB

Trunk Priorities

Trunk priorities of the NCR Century 201 are listed in their order of priority:

    1. Integrated printer, trunk 7
    2. Integrated COT, trunk 0, position 0
    3. Trunk 6
    4. Trunk 5
    5. Trunk 4
    6. Trunk 3
    7. Trunk 2
    8. Trunk 1
    9. Trunk 0

FUNCTIONAL DESCRIPTION

Peripherals connected to the I/O trunks are selected for data transfer by the ALU executing an INOUT command. When the ALU has transmitted all necessary selection characters to the peripheral, the peripheral responds with an end-of-control-information signal, which causes the ALU to store an S-2 status character and proceed with normal internal processing. All data transfer to and from the peripheral is under the supervision of the I/O control.

Data Transfer

In the NCR Century 200, data transfer takes place one character at a time, concurrently with internal processing. Normally, each time a character is ready to be transmitted or received, the ALU flow is interrupted and memory cycles are used by the I/O control to process the transfer of that character. In the NCR Century 201, with the 32-character buffers added to trunks 3 and 4, data transfer from the peripheral to the buffer or from the buffer to the peripheral is not dependent on the interrupt rate and priorities that govern the I/O control when servicing the various trunks. Instead, the speed of data transfer between the buffer and the peripheral is determined by the speed of the peripheral and the trunk.

During input, the I/O control transfers data from the buffer to memory, two bytes at a time, on a first-in, first-out basis. Requests for service from the buffer are processed according to the established trunk priority scheme. If the I/O control is busy and unable to respond to a request within the allotted time, data transfer continues from the peripheral to the buffer; data transfer is not interrupted and the peripheral is not deselected.

As soon as the I/O control becomes free to service the request, it transfers the stored data from the buffer to memory. The design objective of the buffer interface is to prevent overloads by keeping the buffer as empty as possible during data input.

During output, the I/O control transfers data from memory to the buffer, two bytes at a time. From the buffer, data is transferred to the peripheral, one byte at a time, on a first-in, first-out basis. Requests for service from the buffer are processed according to the established trunk priority scheme. If the I/O control is busy and unable to respond to a request within the allotted time, data transfer continues from the buffer to the peripheral; data transfer is not interrupted and the peripheral is not deselected. As soon as the I/O control becomes free to service the request, it transfers additional data from memory to buffer. The design objective of the buffer interface is to prevent overloads by keeping the buffer as full as possible during data output.

Peripheral Restrictions

The only peripherals permitted to be used on the very high-speed trunks (trunk 4 and, optionally, trunk 3), are peripherals that can access the high-speed control-line of the common trunk (for example, certain NCR phase mode tape handlers and certain NCR disc units). All data transfer occurs during the high-speed trunk flows, which are modified to accept data from trunks 3 and 4. (On the NCR Century 200, only trunks 5 and 6 utilize the high-speed trunk flows.)

ENVIRONMENTAL REQUIREMENTS

The trunk interface and other components necessary for conversion to an NCR Century 201 processor are designed to operate within the operating limits of the NCR Century 200 processor, as shown in the following table.

Temperature	$68°$ to $78°$ F Dry Bulb
Humidity	40% to 60% Relative
Altitude	7000 feet maximum