70 | 45-55 TRAINING GUIDE

SPECTRA 70

RADIO CORPORATION OF AMERICA • ELECTRONIC DATA PROCESSING

SYSTEM 70 | 45 55

TRAINING GUIDE

SPECTRA 70

RADIO CORPORATION OF AMERICA • ELECTRONIC DATA PROCESSING

SYSTEMS

70 | 45/55

TRAINING GUIDE

RADIO CORPORATION OF AMERICA

70-45-801

The information contained herein is subject to change without notice. Revisions may be issued to advise of such changes and/or additions.

First Printing: December, 1964
Second Printing: January, 1965

CONTENTS

CONTENTS (Continued)

FOREWORD

70/45-55 TRAINING GUIDE

This manual is designed as an instructor/student guide to the Spectra 70/45-55 Systems. Emphasis is placed on programming techniques and on the application of the more powerful elements of the systems. Many programming illustrations and exercises are supplied. While the manual is organized from a pedagogical point of view, it depends heavily on the 70/45-55 Assembly and System Reference Manuals and should not be considered without access to these documents.

On the matter of content, the basic processor functions and instructions are described. Input/Output operations and programming are described as they relate to the basic processors. Programming is illustrated in the basic assembly language and the features of the basic assembly system are described.

Inexperienced programmers might find it helpful to study either the 70/15 or 70/25 Training Manual before studying this document.

FUNCTIONAL DESCRIPTION OF RCA 70/45-55 SYSTEMS

INTRODUCTION

This section of the manual describes the functional characteristics and the principal features of the 70/45-55 processors. The intent is to give the reader a coherent and meaningful summary of the capabilities of the processor, primarily from a system viewpoint.

This introductory section is not excessively concerned with detail, for each of the topics discussed here will be amplified in the succeeding sections. Appended to this section is a definition of the symbols used throughout the manual.

SYSTEM CHARACTERISTICS

The RCA 70/45-55 are medium- to large-scale processors within the Spectra 70 family.

Some of their salient features are summarized in Charts 1 and 2.

An extremely important feature of the processors is the compatibility that they possess with the other processors in the Spectra 70 family, and with the IBM 360 System. The following charts summarize these compatibility features.

SCRATCH-PAD MEMORY

The Scratch-Pad Memory contains the processor working registers. Each location is addressable and may be manipulated by several of the privileged instructions.

- 4 Processor States for efficient interrupt control
- Unique sets of general-purpose registers for each state
- Privileged instructions (those which can be executed only when specified by a program switch)
- Variable length instructions
- EBCDIC or ASCII Code
- Fixed and variable length data formats
- Memory protection
- Elapsed time clock
- 32 individually maskable interrupt conditions
- Multiplexor channel to simultaneously control up to 8 slow-speed I/O devices
- High-speed selector channels
- Multiplexor and each selector can control up to 256 I/O devices
- Wide variety of I/O devices, mass storage and communication equipment
- 11-way simultaneity on 70/45
- 14-way simultaneity on 70/55
- Up to 256 communications devices can operate simultaneously
- Read/write direct control for multiprocessor systems
- Scratch-Pad Memory of 512 bytes with 300ns 32-bit access cycle time
- DXC (Data Exchange Control) memory-to-memory transfer between 2 processors
- SHARING of memory by processors (70/55 only)
- Floating point arithmetic for scientific applications (optional in 70/45)

- Chart 2 -

	70/45	70/55
Scope	Medium	Medium - Large
Maximum Data Rate	465K bytes	640K bytes
Storage	16-262K bytes	65-524K bytes
Access Time	1.44us	.84us
Data Path	2 bytes	4 bytes
Implementation	Logic controlled by EO's contained in 2 read only memories	Order code implemented by Hardware

- Chart 1 -

Compatibility (within the Spectra family)

A. Data Compatible
   1. Same internal data code used by all processors.
   2. Any peripheral device controller and its associated devices can be operated by any processor.

B. Program Compatible
   1. 70/25 programs executable on 70/45-55 with
      a. Minor editing
      b. Reassembly to link into 70/45-55 Operating System
   2. Two-way compatibility between 70/45 and 70/55.

- Chart 3 -

Compatibility with System 360

A. Machine language compatible with 360 series systems having similar complements for all programs limited to 360's "Problem State".

B. 360 series assembly language programs may be assembled and run on 70/45-55 provided the following are redefined:
   1. Linkage to and descriptive information for 360 I/O and executive control routines.
   2. User defined macro-instructions.
   3. Privileged operations within user's own code.
   4. References to standard locations (first 128 bytes of storage).

C. External control parameters to the operating system must be defined in accordance with 70/45-55 requirements.

- Chart 4 -

Scratch Pad contains:

1. General-purpose registers for each processor state.
2. Floating-point registers.
3. Interrupt masks.
4. Program control information.
5. Control words for I/O operations.
6. Buffer area for I/O operations.
7. Various hardware utility registers.

- Chart 5 -

Scratch Pad Characteristics:

- 128 Words
- 32 bits per word
- 300 nanosecond cycle time
- Locations are uniquely addressed
- Not part of main memory

- Chart 6 -

GENERAL-PURPOSE REGISTERS

Each program state has its own set of general-purpose registers. These registers can be used:

1. as index registers
2. in address arithmetic and indexing
3. as accumulators for fixed point and logical operations.

Each register has a 32-bit capacity and is designated by number, 0-15. For some operations, two adjacent registers can be coupled, providing a two-word capacity.

FLOATING-POINT REGISTERS (optional in 70/45)

Four floating-point registers are provided having a 64-bit capacity. They are designated by the numbers 0, 2, 4, and 6.

DATA FORMATS

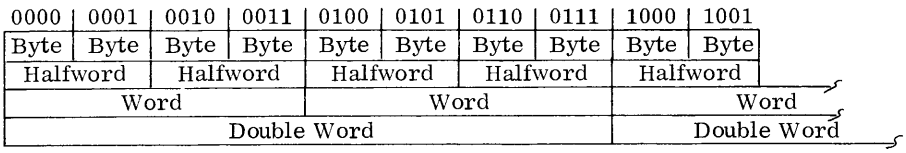
The basic building block of the various data (and instruction) formats is the byte. A byte contains 9 bits--8 bits of information plus a parity bit, which is used for accuracy control.

Bytes may be grouped to form:

1. Halfwords - 2 consecutive bytes
2. Words - 4 consecutive bytes
3. Double Words - 8 consecutive bytes

Each byte is binarily addressable starting with location 0. Fixed length fields such as halfwords or words must begin on integral boundaries as shown in the following chart.

Binary Address



- Chart 7 -

Note, therefore, that the addresses for the various units of information are related in the following way:

Unit	Address Must Be a Multiple Of
Byte	1
Halfword	2
Word	4
Double Word	8

- Chart 8 -

Fixed length data of 8, 16, 32, or 64 bits and variable length data of up to 256 characters may be processed. The eight data formats are summarized below.

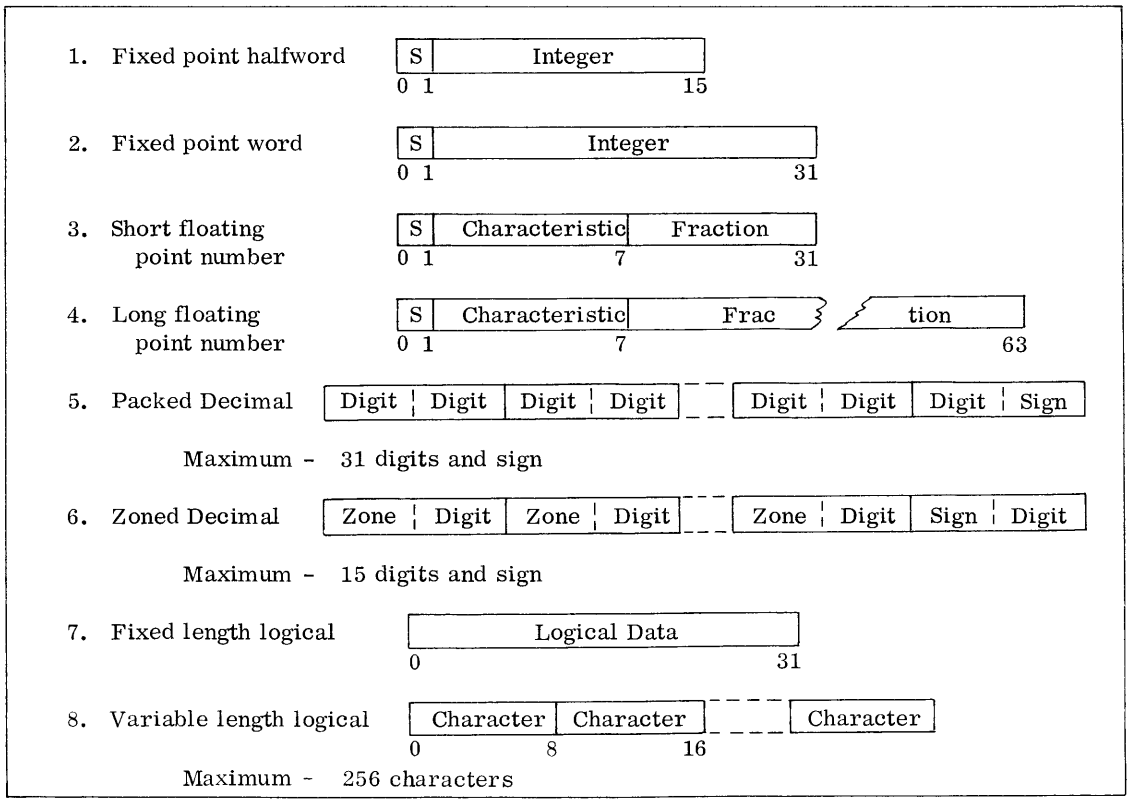
CHARACTER CODE

Characters(8-bit bytes)may be represented in either EBCDIC or ASCII code with equal facility. The user specifies the code by setting a program switch.

Illustration

The characters 'RCA' are shown in both codes:

ASCII	1011 0010	1010 0011	1010 0001
	R	C	A
EBCDIC	1101 1001	1100 0011	1100 0001



Illustration

The number +1234 is expressed in the 4 possible decimal formats.

1. Zoned-Decimal, EBCDIC

1111 0001	1111 0010	1111 0011	1100 0100
1	2	3	+ 4

2. Packed-Decimal, EBCDIC

0000 0001	0010 0011	0100 1100
1	2 3	4 +

3. Zoned-Decimal, ASCII

0101 0001	0101 0010	0101 0011	1010 0100
1	2	3	+ 4

4. Packed-Decimal, ASCII

0000 0001	0010 0011	0100 1010
1	2 3	4 +

INSTRUCTION FORMATS

There are five basic instruction formats. The instruction format expresses, in general terms, the operation to be performed as shown in the following chart:

RR	register-to-register operation
RX	register-to-indexed storage operation
RS	register-to-storage operation
SI	storage-and-immediate operand operation
SS	storage-to-storage operation

- Chart 9 -

The detailed format of each instruction is shown in Chart 10. (See following page.)

The instruction sub-fields are defined as follows:

R_1, R_2, R_3	4 bit operand register specification
X_2	4 bit index register specification
B_1, B_2	4 bit base register designator
D_1, D_2	12 bit displacement
I_2	8 bit immediate operand
L_1, L_2	4 bit operand length specification
L	8 bit operand length specification

- Chart 11 -

ADDRESS GENERATION

The effective storage address is computed from the following binary components:

1. Base (contents of the designated base register, B_1 or B_2)
2. Displacement (D_1, D_2), and
3. Index (contents of the designated index register, X_2) for RX instructions

In computing the address, the base and index are treated as unsigned 24-bit positive binary integers in bits 8-31 of the designated register. The displacement is treated as a 12-bit positive binary integer. The effective address is computed by adding the components as binary numbers, ignoring overflow.

NOTE: If register 0 is specified as the base register and/or the index register, then a zero quantity is to be added, regardless of the contents of register 0.

The following illustrations use decimal addresses for convenience.

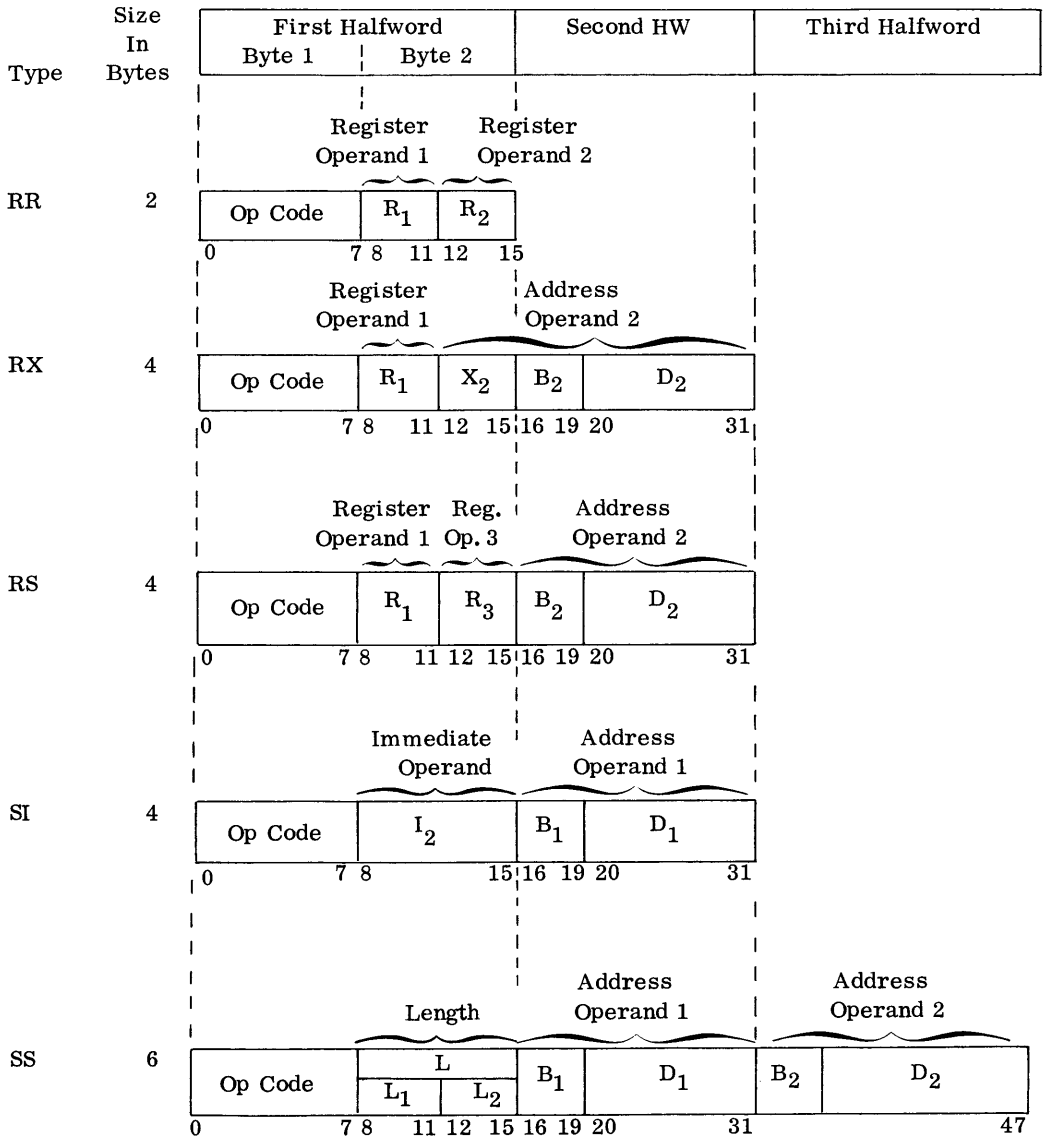
Illustration

Assume Reg. 12 00 00 20 00
Reg. 13 00 00 04 80

Instruction	B_1	D_1	B_2	D_2
	12	0	13	20

The effective address of B_1, D_1 is 00 00 20 00.

The effective address of B_2, D_2 is 00 00 05 00.


Type
Size In Bytes
First Halfword
Byte 1
Byte 2
Second HW
Third Halfword
RR
2
Register Operand 1
Register Operand 2
Op Code
R1
R2
0 7 8 11 12 15
RX
4
Register Operand 1
Address Operand 2
Op Code
R1
X2
B2
D2
0 7 8 11 12 15 16 19 20 31
RS
4
Register Operand 1
Reg. Op. 3
Address Operand 2
Op Code
R1
R3
B2
D2
0 7 8 11 12 15 16 19 20 31
SI
4
Immediate Operand
Address Operand 1
Op Code
I2
B1
D1
0 7 8 15 16 19 20 31
SS
6
Length
Address Operand 1
Address Operand 2
Op Code
L
L1
L2
B1
D1
B2
D2
0 7 8 11 12 15 16 19 20 31 47

- Chart 10 -

Illustration

Assume Reg. 3 00 03 00 10
Reg. 2 00 00 02 00

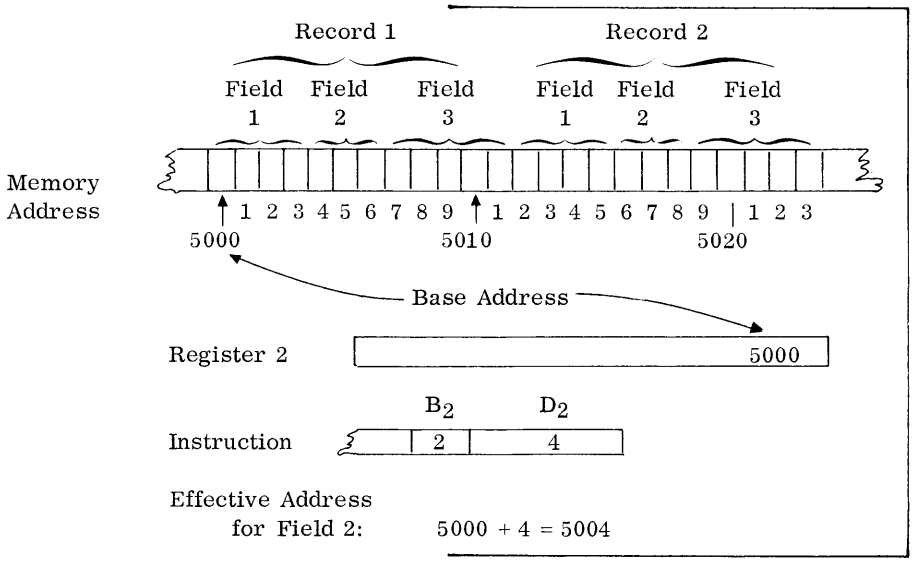
Instruction X_2 B_2 D_2

3	2	1000

The effective address, composed of X_2, B_2, D_2, is

00 03 00 10 + 00 00 02 00 + 1000 = 00 03 12 10.

Illustration



To process Record 2, the contents of Register 2 are increased by 12, the size of Record 1. The instruction field illustrated would then refer to Field 2 of Record 2.

If the instruction is an index-type instruction, a second general-purpose register, called an index register, also generates the address. In the above illustration, the index register (Register 3, for example) would initially be cleared to zero. After the first record is processed, the contents of Register 3 is increased by the record size. Then the instructions using Registers 2 and 3 for addressing would refer to Record 2. An illustration of this follows:

Register 2 (base) 5000

Register 3 (index) 12

Instruction X_2 B_2 D_2

3	2	4

Effective Address for Field 2,
Record 2: 5000 + 12 + 4 = 5016

PROGRAMMING NOTE ON ADDRESSING

Because overflow is ignored, "wrap-around" can occur. The RCA 70/45 System allows addressing of a maximum of 262,144 bytes beyond which "wrap-around" occurs. The 70/55 allows for addressing of a maximum of 524,288 bytes before "wrap-around" occurs.

Illustration

Assume B_2 262 143
D_2 000 003, then the

70/45 effective address is 000 002 and the 70/55 effective address is 262 146.

The address wrap-around property provides negative indexing.

Illustration

It is desired to decrement the storage address by 1.

Thus, the rightmost 24 bits of the register specified by X_2 should be all 1 bits; i.e., 524,287.

In forming constants to get deliberate wrap-around, say for decrementation, the situation should be viewed as if both processors wrapped at 524,288 to maintain compatibility. Thus, in the preceding example, although decrementation could be achieved on the 70/45 with X_2 = 262,143, such would not be the case in the 70/55.

INSTRUCTION EXECUTION

Instructions are performed by the basic processing unit primarily in the sequential order of their locations. This sequence is altered by branching instructions and program interrupts.

The detailed operation of the various branching instructions is determined by the Condition Code (CC). The two bits of the Condition Code provide for four possible settings--0, 1, 2, and 3. The Condition Code reflects such conditions as first operand high, equal, overflow, channel busy, etc.

PROGRAM INTERRUPT

The interrupt system is designed to respond to asynchronously occurring external and I/O signals and to monitor exceptional conditions generated by the program Processor. The interrupt is facilitated in that the processor has four program states, each with its own set of registers.

Processor States

1. Processing (User's program(s))
2. Interrupt response (interrupt conditions serviced)
3. Interrupt control (interrupt conditions analyzed)
4. Machine condition (machine error interrupts)

- Chart 12 -

There are thirty-two possible interrupt conditions such as Op-code trap, decimal overflow, selector trunk 1 terminate, etc., each individually maskable. Except for machine error conditions, if an interrupt condition is effected and is not masked out, control is given to state 3.

Other important features facilitating the interrupt system are outlined in Chart 13.

Interrupt Features

A. Read/Write Direct Control
B. Privileged instructions.
C. 32 interrupt conditions, each individually maskable.
D. Priority scheme for interrupt conditions.
E. Weight register allows cause of interrupt to be detected efficiently.
F. Elapsed time clock with maximum cycle of 15.5 hours.
G. Memory protection in blocks of 2048 bytes.

- Chart 13 -

INPUT/OUTPUT

A wide variety of input/output devices may be connected to the 70/45-55 processors. The processors contain I/O channels so that these devices can run off line, independent of program operation.

There are two types of I/O channels: Selectors and Multiplexors. A selector channel may have up to 256 devices connected, but only one may be operating at a time. Each of these devices is connected to a Control Electronics which, in turn, connects to the Selector via the RCA Standard I/O Interface. In general, several devices may be connected to a Control Electronics.

A Multiplexor channel may have connected up to 256 devices, and it is possible for all the devices to be operating at one time. The Multiplexor may be operated in a burst mode so that the Multiplexor in effect functions as a Selector.

Diagram 1 depicts the logical connection of the I/O channels and I/O devices. (See following page.)

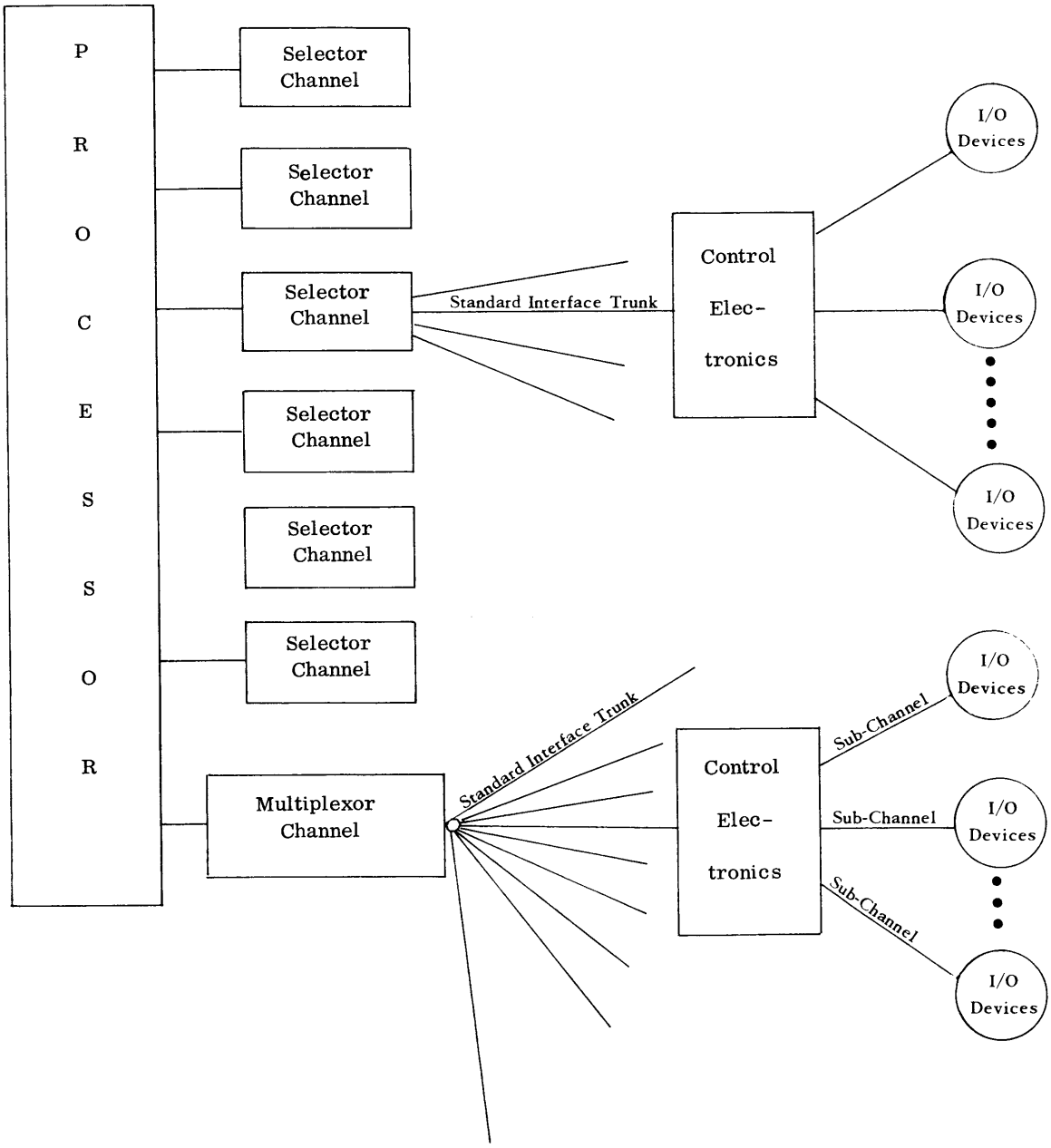
The following chart summarizes the I/O characteristics of the two processors.

I/O Characteristics	70/45	70/55
Maximum data rate	465K bytes	640K bytes
Number of selectors	0-3	0-6
Control Electronics/selector	1-2	1-4
Devices/selector (maximum)	256	256
Multiplexor (standard)	1	1
Control Electronics for Multiplexor	8	8
Devices attached to Multiplexor	256	256
Maximum total Control Electronics	14	20

- Chart 14 -

- Diagram 1 -

70/45-55 Input/Output Flow



Some possible I/O connections are shown below.

I/O Connections

70/45	# Control Electronics Attached Per Channel			
	A	B	C	D
Selector 1			2	2
2			2	2
3				1
Multiplexor	1	8	1	8

- Chart 15 -

Thus, in configuration D, Chart 15, 11-way simultaneity is possible because each of the 3 selectors can control an I/O device while the Multiplexor can control 8 devices at the same time.

70/55	# Control Electronics Attached Per Channel			
	A	B	C	D
Selector 1		2	4	4
2			2	2
3			1	2
4			1	2
5				1
6				1
Multiplexor	1	8	4	8

- Chart 16 -

Because devices operating in the multiplex mode require more processor time to service each byte than selector or burst mode, the peak data rates must be weighted. The weights for the various modes of operation are shown below.

	70/45 Weight	70/55 Weight
Multiplexor	7.5	4.0
Burst Mode	1.0	1.0
Selector	1.0	1.0

Thus, two 120KC tapes could not operate simultaneously via the Multiplexor because their combined data rate, 960KC, exceeds the maximum data rate of the processor.

SYMBOLOGY

The following symbols are used throughout this manual to define and describe instructions.

Bn	Instruction base register number field n (B_1 or B_2)
Dn	Instruction displacement field n (D_1 or D_2)
I_2	Immediate data byte of an instruction (operand 2)
Ln	Instruction length-of-operand field n (L, L_1, or L_2)
M	Instruction mask field (replaces an R field)
Rn	Instruction general-purpose register field n (R_1, R_2, or R_3)
(Rn)	The contents of the general-purpose register specified
$\lvert Rn \rvert$	The absolute value of the contents of Rn
R_n+1	The next sequential register following the one specified
RN	General-purpose register N; N=0, 1, 2,..., 15
RN+1	The next sequential register following RN
S	Storage field of an assembly instruction (S, S_1, or S_2)
Sbn	Storage location whose address is (Bn)+Dn
Srn	Storage location whose address is contained in Rn
Sxn	Storage location whose address is (xn)+(Bn) + Dn
X_2	Instruction index register field (operand 2)
$\longrightarrow$	Is transferred to
=	Is equal to
>	Is greater than
<	Is less than
$\leq$	Is less than or equal to
$\geq$	Is greater than or equal to
$\neq$	Is not equal to

Note that symbols such as Bn represent some base (general-purpose) register and/or its address as an instruction field. Also, while Sbn, for example, is the address of one byte in memory, it also refers to the entire field starting at that byte.

Questions

1. How many general-purpose registers are there in the Processing State? What are their uses?

2. To what do ASCII and EBCDIC refer?

3. Match the following:

ASCII	_____	a.	32 bits
Packed	_____	b.	Zoned data
Word	_____	c.	8 bits
Byte	_____	d.	Decimal data

4. What is the result of packing the following field? Is it plus or minus?

11111001	11110000	11110101	11010110

5. What are the sizes of the following numeric fields?

   a. fixed-point number

   b. floating-point number

   c. decimal arithmetic field

6. What is integral storage alignment?

7. What is the Processing State?

8. What is the Condition Code?

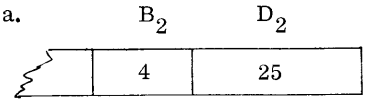
9. What is immediate data?

10. General-purpose registers 2, 3, 4, and 5 contain the following addresses:

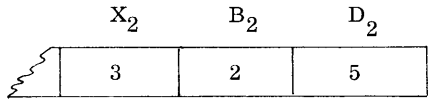
2	3000
3	25
4	4000
5	142

What are the effective addresses generated for the following instruction fields?

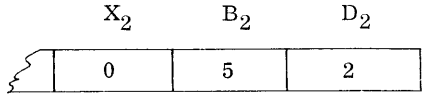
a.



b.



c.



11. What are the major differences between selector and multiplexor channels?

DECIMAL ARITHMETIC

This section describes the instructions and data format associated with decimal arithmetic. Instruction formats, in machine and written form, are illustrated. Overflow and the condition codes set by the instructions are illustrated.

INSTRUCTION FORMAT

SS form:

Bytes	1	2		3	4	5	6
	OP	L_1	L_2	B_1	D_1	B_2	D_2

Written form: OP $D_1(L_1, B_1), D_2(L_2, B_2)$.

Note that the written L's are one unit greater than the machine L's; i.e., the written L is the actual field length, the machine L is one byte less.

RX form:

Bytes	1	2		3	4
	OP	R_1	X_2	B_2	D_2

Written form: OP $R_1, D_2(X_2, B_2)$

DATA FORMAT

Decimal operations use the packed format for data. Z means zone in the following example.

Unpacked: Z1 Z2 Z3 +4

Packed: 01 23 4+

PACKING, UNPACKING DATA

PACK(SS):	Delete zones, pack $(Sb_2) \rightarrow Sb_1$	$D_1(L_1, B_1), D_2(L_2, B_2)$	R
UNPK(SS):	Insert zones, unpack $(Sb_2) \rightarrow Sb_1$	$D_1(L_1, B_1), D_2(L_2, B_2)$	R

(Sb_2) is extended with leading zeros, if necessary, to fill Sb_1. (The R indicates that the operation is performed right to left.) The fields may overlap.

DECIMAL OPERATIONS

When CC appears following an instruction definition, it indicates that the Condition Code is set. The L indicates a left to right operation. (See table below.)

Note that the MVC instruction specifies only one L field. Thus, L can have a maximum value of 256 (255 in machine form).

Programming note: Because the move operation is performed from left to right, one character at a time, it is possible to propagate a character through a field by having the Sb_1 field start one byte to the right of the Sb_2 field. The two fields must overlap.

CONDITION CODES

OPERATION	0	1	2	3
Arithmetic result is	0	⊖	+	overflow
Operand 1 compares	=	<	>	

Note that overflow only applies to AP, SP, and ZAP. Also, plus zero compares as equal to minus zero.

Name (Form)	Mnemonic	Operation	Written Operand Form	
Add Decimal (SS)	AP	$(Sb_1) + (Sb_2) \rightarrow Sb_1$	$D_1(L_1, B_1), D_2(L_2, B_2)$	CC
Subtract Decimal (SS)	SP	$(Sb_1) - (Sb_2) \rightarrow Sb_1$	$D_1(L_1, B_1), D_2(L_2, B_2)$	CC
Zero and Add (SS)	ZAP	$0 + (Sb_2) \rightarrow Sb_1$	$D_1(L_1, B_1), D_2(L_2, B_2)$	CC
Divide Decimal (SS)	DP	$(Sb_1) \div (Sb_2) \rightarrow Sb_1$	$D_1(L_1, B_1), D_2(L_2, B_2)$	
Multiply Decimal (SS)	MP	$(Sb_1) \times (Sb_2) \rightarrow Sb_1$	$D_1(L_1, B_1), D_2(L_2, B_2)$	
Compare Decimal (SS)	CP	Compare $(Sb_1) : (Sb_2)$	$D_1(L_1, B_1), D_2(L_2, B_2)$	CC
Move (SS)	MVC	$(Sb_2) \longrightarrow (Sb_1)$	$D_1(L, B_1), D_2(B_2)$	L
Branch on Condition (RX)	BC	Branch to Sx_2 if a mask bit corresponds to a condition code.	M, $D_2(X_2, B_2)$	

Illustrations

The relation of the Condition Code and several masks of a BC instruction is shown below:

CC	0	1	2	3	
M=4	0	1	0	0	test CC1
M=12	1	1	0	0	test CC0 or 1
M=15	1	1	1	1	test all

When M = 15 the instruction acts as an Unconditional Branch instruction.

MULTIPLICATION

(Sb_1) 00 00 42 34 1+ (L1=5) x (Sb_2) 01 2- (L2=2)

Product 00 05 08 09 2-

Invalid Operations

(Sb_1)	OP	(Sb_2)	Reason
42 3+	+	62 1+	Sum too large
42 3+	+	24 00 0-	(Sb_2) too long
02 5+	x	00 6-	$L_2 = L_1$
63 4+	x	4-	(Sb_1) lacks leading zeros
41 24 0+	÷	52 3-	(Sb_1) lacks leading zero
00 41 0+	÷	03 2-	0320 < 0410 the quotient would be too large
00 41 2+	÷	05 23 0-	$L_2 = L_1$

Instruction Illustration

Locations 4004-4007: 00 00 24 3+ Sb_1
Locations 4008-4009: 92 5+ Sb_2
Register 4: 40 00

Written form: OP D_1 (L_1, B_1), D_2(L_2, B_2)
MP 4(4, 4), 8(2, 4)

Result 4004-4007: 02 24 77 5+

Example 1: (Z represents the four-bit zone)

Item	Data Form	Value	Locations
Part Number	Zoned	Z2 Z3 Z4	4001-4003
Quantity	Packed	01 24 3+	4004-4006
Unit Cost ($9.25)	Packed	92 5+	4007-4008
Discount factor	Packed	09 0+	4009-4010
Test constant	Zoned	Z5 Z0 Z0 +0	4011-4014

Prepare an output item, in zoned form, containing the part number and total cost, starting at location 5000. If the quantity equals or exceeds 5000, apply a 10% discount to the total cost. Assume that General-Purpose Registers 1, 4, and 5 contain 1000, 4000, and 5000 respectively.

DIVISION

(Sb_1) 04 12 4+ (L1=3) ÷ (Sb_2) 52 3- (L2=2)

7- 46 3+

Quotient: L_1-L_2 bytes
Remainder: L_2 bytes

Examples of valid and invalid arithmetic operations follows:

Valid Operations

(Sb_1)	OP	(Sb_2)	Result: (Sb_1)
42 3+	+	00 02 4-	39 9+
00 08 9+	×	0-	00 00 0-
00 4+	×	4-	01 6-
04 12 4+	÷	52 3-	7 - 46 3+ (rem.)
00 41 2-	÷	32 3+	1 - 08 9- (rem.)

(SS operand form: $D_1(L_1, B_1), D_2(L_2, B_2)$)

Storage Location	Inst. Bytes	OP	Operand	Remarks
1000	6	MVC	0(3,5), 1(4)	Part number ⟶ output
1006	6	PACK	15(3,4), 11(4,4)	Pack the test constant
1012	6	CP	4(3,4), 15(3,4)	Quantity: 5000
1018	4	BC	10, 44(0,1)	If Qty ≥ 5000, go to 1044
1022	6	ZAP	15 (5,5), 4(3,4)	Extend quantity field
1028	6	MP	15 (5,5), 7(2,4)	Quantity x cost = total cost
1034	6	UNPK	3(7,5), 15(5,5)	Unpack total cost
1040	4	BC	15, 40(0,1)	Go to 1040 (wait)
1044	6	ZAP	15(7,5), 4(3,4)	Extend quantity field
1050	6	MP	15(7,5), 7(2,4)	Quantity x unit cost
1056	6	MP	15(7,5), 9(2,4)	Apply discount
1062	6	UNPK	3(9,5), 15(7,5)	Unpack total cost

ANALYSIS OF CODING

Instruction Location	
1000	(4001-4003) transferred to 5000-5002
1006	(4011-4014) packed and transferred to 4015-4017
1012	(4004-4006) compared to (4015-4017)
1018*	Mask of 1010 compared to Condition Code. Instruction branches if code is 0 or 2, equal or greater.
1022	Four leading zeros are added to the number from 4004-4006 so that it is large enough to contain the product.
1028	(5015-5019) multiplied by (4007-4008)
1034	Zones inserted in number in 5015-5019, expanding it to 5003-5009
1040*	An unconditional branch to the location containing the branch instructions has the effect of stopping the computer
1044-1062	These instructions are executed when the quantity equals or exceeds 5000. Note that 90% of the total cost produces a 10% discount and a result with four decimal places.

*When zero is used for an X or B field, the field is ignored. Register zero is not specified.

Example 2:

Locations	Contents
1000-1001	N1 02 5-
1002-1007	N2 00 00 05 73 42 7+
1008-1010	N3 24 32 6+
1011	N4 5+
1012-1013	N5 01 2+
Register 2	5 00
Register 3	10 00

Calculate $R = \frac{N1\ (N2-N3)}{N4 + N5}$

Storage Location	OP	Operand	Results	
0500	SP	2(6, 3), 8(3, 3)	(1002-1007):	00 00 05 49 10 1+
0506	MP	2(6, 3), 0(2, 3)	(1002-1007):	00 01 37 27 52 5-
0512	AP	12(2, 3), 11(1, 3)	(1012-1013):	01 7+
0518	DP	2(6, 3), 12(2, 3)	(1002-1007):	08 07 50 1- 00 8-

Exercise 1

Write the instructions to propagate the byte in location 5000 through the field in location 4000-4055. Assume that register three contains 3000.

Exercise 2

Input data (Zoned)	Locations	Format
Part number	4000-4007	ZX ZX ZX ZX ZX ZX ZX ZX
Quantity	4008-4011	ZX ZX ZX +X
Unit cost	4012-4015	ZX ZX ZX +X

assumed decimal point

Constants (Packed)	Locations	Format
95%	5000-5001	09 5+
90%	5002-5003	09 0+
+1000	5004-5006	01 00 0+
+5000	5008-5010	05 00 0+
+100	5012-5013	10 0+
+2	5014	2+

Registers	Contents
2	2000
4	4000
5	5000
6	6000

Prepare an output item in packed form starting at location 6000 and consisting of the part number and total cost. The total cost is computed as follows: if quantity exceeds 5000, the first 1000 units receive a 10% discount, half of the remainder receives a 5% discount, and the remaining half are billed at the unit cost. (Assume that the quantity is an even number.) If the quantity does not exceed 5000, it is billed at the unit cost. Locate the assumed decimal point in both possible results.

This example may be shown symbolically as:

If $Q > 5000$, total cost =

$$1000 \times U \times 90\% + \frac{Q-1000}{2} \times U \times 95\% + \frac{Q-1000}{2} \times U$$

If $Q \leq 5000$, total cost = $Q \times U$

(Note: Keep track of the position of the assumed decimal point throughout. The factor of 100 will be required to adjust the number of decimal places in at least one partial result.)

RCA 70/45-55 ASSEMBLY SYSTEM

This section introduces the basic features of the RCA 70/45-55 Assembly System. The use of symbols and expressions is illustrated. The method of defining storage and constants is explained, and the details of writing instructions are described.

PROGRAM FORMAT

The elements of a program are illustrated by the following chart.

	NAME	OPERATION	OPERAND	COMMENTS
	Symbol	Mnemonic Instruction Code	Symbols, Expressions	
MACHINE STATEMENTS	START	AP	ABLE(3), BAKER	ABLE+BAKER
		BC	15, *+14	LOCATION COUNTER
	Symbol	Assembly Instruction	Defining Fields	
ASSEMBLY STATEMENTS	ABLE	DS	2C24	DEFINE STORAGE
	CONST	DC	2L3C'123'	DEFINE CONSTANT
	REGL	EQU	10	EQUATE SYMBOLS

SYMBOLS

Symbols are prepared by the user to identify statements (names), work areas, and I/O units. Examples of valid and invalid symbols follow.

Valid	Invalid
A	1A (begins with a decimal digit)
A12	A+23 (contains a special symbol)
ABLE2A	ABLEABLEABLE (too long)
	A BLE (contains a blank)

Symbols are either absolute or relocatable.

The EQU instruction can define a symbol and make it absolute or equate one symbol with another. Examples of absolute and relocatable symbols follow. It is assumed that all of the EQU instructions below are executed in order in the same program.

NAME	OPERATION	OPERAND	
REGL	EQU	1	Absolute
OPND	EQU	REGL	Both absolute
BEGIN	BC	2, 3(4, 5)	Relocatable
START	EQU	BEGIN	Both Relocatable

EXPRESSIONS

Expressions specify operand fields of machine and assembly instructions. They may be simple or compound as shown below:

SIMPLE EXPRESSIONS

ABLE(symbol)
2, C'B' } self-defining values
*(Location counter)

COMPOUND EXPRESSIONS

*+23
ABLE +10
ABLE-BAKER+ALPHA
A+B*Y=(A+(B*Y))

(Self-defining values may be decimal or hexadecimal numbers up to six digits in length, or a single character.)

INVALID COMPOUND EXPRESSIONS

		Correct Form
ABLEX'02'	(arithmetic operator missing)	ABLE*X'02'
A+-10	(two arithmetic operators together)	A-10
-ALPHA+10	(begins with arithmetic operator)	ALPHA+10
A-B+C+10	(too many terms)	A-B+C
**5	(multiply follows location counter)	*+5

Expressions are either relocatable or absolute depending on the nature and relation of the included symbols. A relocatable expression must be positive. An absolute expression may be negative.

In the illustrations below, the symbols have the following values:

A	2000	relocatable
B	1000	relocatable
C	4000	relocatable
FOX	500	absolute

ABSOLUTE EXPRESSIONS	VALUE
A-B+10	1010
FOX*10	5000
FOX+25	525
A-B+FOX	1500

RELOCATABLE EXPRESSIONS	VALUE
A-FOX	1500
*+10	Depends on location counter
A+10	2010
A-B+C	5000

Note that the value of a relocatable symbol or expression depends on where the program is loaded. The maximum value of an expression is 131,071.

INVALID EXPRESSIONS

		Correct Form
A+B+10	(two relocatable symbols added)	A-B+10
A*FOX	(relocatable symbol multiplied)	FOX*5
A+B+C	(no minus)	A-B+C
A-B-C	(no plus)	A-B+C
FOX-A	(relocatable subtracted from absolute)	FOX+A
B-2000	(relocatable expression cannot be negative)	B-FOX

DEFINING STORAGE

Input/Output areas and work areas are reserved by the Define Storage instruction, DS. The form of its operand is dCLn. (d is the duplication factor and n is the number of bytes in the length of storage before duplication.) In the following example, 10 bytes (characters) of storage will be reserved starting at ALPHA, ten starting at BETA, and ten at GAMMA.

NAME	OPERATION	OPERAND
ALPHA	DS	2CL5
BETA	DS	CL10
GAMMA	DS	10C

Note that the storage area reserved is not cleared. Ln has a maximum value of 256.

Exercise 1

NAME	OPERATION	OPERAND
COST	DS	10CL2
FILE	DS	80C
WORK	DS	10CL10
CON	EQU	125
MAST	EQU	FILE-COST

Assume that the above instructions are executed. Indicate which of the following symbols and expressions are valid, invalid, absolute, or relocatable.

a. MAST+CON
b. COST*CON
c. COST-CON+FILE
d. MAST*CON
e. WORK+CON-COST

DEFINING CONSTANTS

Constants are defined by the Define Constant instruction: DC. Its form is dCLn 'characters'. The following instructions cause the constant 123bb123bb to be created (b = blank) in zoned form.

NAME	OPERATION	OPERAND
CON1	DC	2CL5'123'
CON2	DC	C'123bb123bb'

Note that if Ln is too small, the rightmost characters of the constant within the quotation marks are truncated. The maximum length of a character constant is 16 bytes, before the duplication factor is applied.

ATTRIBUTES

Symbols and expressions generally have two attributes: a value and a length. The value may be a storage address, a register number, or a numerical value. The length attribute is used by the assembly system to provide L fields for SS instructions. The following illustration assumes that the coding is assembled according to the storage locations shown on the left. The attributes apply to the expressions or symbols on the same line. (See example on following page.)

Note that the length attribute of a compound expression is the implied length of its first simple expression (e.g. ABLE in ABLE+10 above). If the first simple expression is a self-defining value, the length attribute of the expression is one (10+ABLE above).

Storage Location	NAME	OPERATION	OPERAND	Attribute Value	Attribute Length
1000	ABLE	DS	20CL3	1000	3
1060	BAKER	DS	6C	1060	1
1066	CONST1	DC	2C'123'	1066	3
1072	CONST2	DC	4CL10'123'	1072	10
	REG	EQU	2	2	1
1112	BEGIN	LR	3,4	1112	2
1114		AP	ABLE+10,3(2,4)	1010	3
1120		AP	10+ABLE(3),4(1,5)	1010	1

STATEMENT FIELDS

An assembly program consists of a sequence of statements. A statement is composed of four fields, each separated by at least one blank: Name field, Operation field, Operand field, and Comment field.

NAME FIELD

The Name field is blank unless it is necessary to reference the statement, in which case a symbol is written. A comment may also begin in the Name field as illustrated on page 19.

OPERATION FIELD

The Operation field contains a mnemonic machine or assembly system instruction.

NAME	OPERATION	
	START	assembly instruction
ADD	AP	machine instruction
ABLE	DS	assembly instruction

OPERAND FIELD

The Operand field contains information to complete the instruction begun in the Operation field.

MACHINE INSTRUCTION OPERANDS

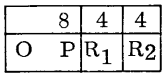
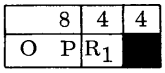
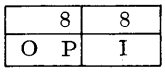
Instruction Type	Explicit Form	Examples
RR	R_1, R_2	2, REG3
RX	$R_1, D_2(X_2, B_2)$	TEN, 25(2, 3)
RS	$R_1, R_3, D_2(B_2)$	1, 2, 125(3)
SI	$D_1(B_1), I_2$	40(9), C'$'
SS	$D_1(L_1, B_1)$, $D_2(L_2, B_2)$	10(3, 1), 5(4, 2)

Instruction Type	Implied Base Register Form	Implied Base Register Examples
RX	$R_1, S_2(X_2)$	REG2, ABLE(3)
RS	R_1, R_3, S_2	3, 4, ABLE
SI	S_1, I_2	ABLE, C'B'
SS	$S_1(L_1), S_2(L_2)$	ABLE(5), BAKER

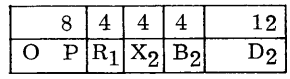
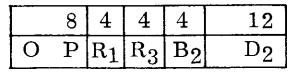
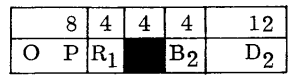
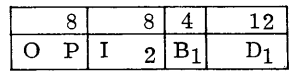
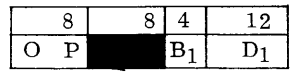
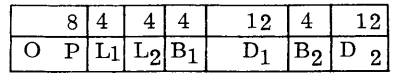
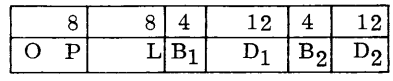
Note that storage fields (S_1, S_2) are relocatable expressions. All other fields must be absolute. The Assembly System determines the B and D fields for ABLE, and the B_1, D_1, and L fields for BAKER. BAKER must have a suitable length attribute, which would have been assigned by a Define Storage instruction, for example; note also that the Assembly L is one byte longer than the machine L. (The Chart on the following page shows all possible combinations for operand fields.)

In order to write machine instructions with implied base registers, a procedure utilizing the USING assembly instruction must be employed. The method will be explained in a later section. The programmer must keep in mind that general purpose registers must be loaded with addresses bracketing the entire area of memory required. For example, assume that the programmer loaded registers 2 and 3 with the addresses 1000 and 5096 (1000 + 4096), respectively. The assembly system will treat the following instruction operands on the left, as if they were written as shown on the right.

Storage Location	NAME	OPERATION	OPERAND	Converted Operand
1046	BEGIN	AP	ABLE(5), BAKER	904(5, 3), 1304(5, 3)
.		.		
.		.		
.		.		
.		.		
2000		BC	15, BEGIN	15, 46(0, 2)
.		.		
.		.		
.		.		
6000	ABLE	DS	100CL4	
6400	BAKER	DS	14CL5	

MACHINE FORMAT	ASSEMBLER OPERAND FIELD FORMAT	OPERAND FIELD CHARACTERISTICS				
		FIELD	BIT SIZE	SIMPLE COMPOUND	ABS. REL.	MAX. VALUE
RR 	R_1, R_2	R_1, R_2, R_3	4	S	A	15
		D_1, D_2	12	SC	A	4095
		B_1, B_2	4	S	A	15
	R_1	X_2	4	S	A	15
		L_1, L_2	4	S	A	16
		L	8	S	A	256
	I	I, I_2	8	S	A	255
		S_1, S_2	16	SC	R	

MACHINE FORMAT — OPERAND FORMAT

	EXPLICIT BASE, EXPLICIT LENGTH	EXPLICIT BASE, IMPLIED LENGTH	IMPLIED BASE, EXPLICIT LENGTH	IMPLIED BASE, IMPLIED LENGTH
RX 	$R_1, D_2(X_2, B_2)$		$R_1, S_2(X_2)$ R_1, S_2 (no indexing) R_1, D_2 (no X or B)	
RS 	$R_1, R_3, D_2(B_2)$		R_1, R_3, S_2	
	$R_1, D_2(B_2)$		R_1, S_2	
SI 	$D_1(B_1), I_2$		S_1, I_2	
	$D_1(B_1)$		S_1	
SS 	$D_1(L_1, B_1)$, $D_2(L_2, B_2)$	$D_1(, B_1)$ $D_2(, B_2)$	$S_1(L_1)$ $S_2(L_2)$	S_1 S_2
	$D_1(L, B_1)$ $D_2(B_2)$	$D_1(, B_1)$	$S_1(L)$ S_2	S_1

Note that if indexing is not to occur for an RX instruction, zero must be written for the X field if a B field exists. If the field or fields within parentheses are zero, the parentheses and the included field or fields may be omitted. Whenever zero is specified for an Z or B field, the field is ignored.

Illustrations

BC	15,START	No indexing, implied B
SLL	2,15	B omitted (a shift left)
BC	15,25(0,3)	No indexing, B required

Exercise 2

An 80-column card is to be read into storage. It is composed of the following fields:

Field	Name	Columns
1	ID	1-10
2	COST	11-20
3	QTY	21-25
4	LEAD	26-30
5	UNKN	31-80

Define each field, and an 80-byte area called WORK. Write the instructions to transfer all 80 characters to WORK. Multiply COST times QTY a) from the original storage area and b) from their locations in WORK. (Assumes that COST has a sufficient number of leading zeros.)

Exercise 3

What are the length attributes of the DS symbols and valid expressions in Exercise 1?

Comments

The two forms of comments are shown below.

NAME	OPERATION	OPERAND	COMMENTS
*THIS COMMENT		REQUIRES AN ASTERISK	
	BC	2,A12 THIS FOLLOWS THE OPERAND	

Example

Example 1 of page 12 could be coded as shown below. The test factor, 5000, has been defined by a DC instruction. The following facts are significant and should be noted.

MVC

OUTPUT has a length attribute of 1. Thus the length of the field moved (3) must be written. If OUTPUT were defined as 4CL3, the length of field could be omitted.

NAME	OPERATION	OPERAND	COMMENTS
PART	DS	CL3	
QTY	DS	CL3	
COST	DS	CL2	
FACTOR	DS	CL2	
CONST	DC	C'5000'	
OUTPUT	DS	12C	
BEGIN	MVC	OUTPUT(3),PART	PART→OUTPUT
	PACK	CONST,CONST	Z5Z0Z0S0→0005000+
	CP	QTY,CONST	COMPARE QTY:5000
	BC	10,DISCNT	GO TO DISCNT IF ≥
	ZAP	OUTPUT+3(4),QTY	EXTEND QTY WITH 0'S
	MP	OUTPUT+3(4),COST	QTYxCOST→OUTPUT+3
	UNPK	OUTPUT+3(7),OUTPUT+3(4)	UNPACK RESULT
	BC	15,*	WAIT
DISCNT	ZAP	OUTPUT+3(5),QTY	EXTEND QTY WITH 0'S
	MP	OUTPUT+3(5),COST	QTYxCOST→OUTPUT+3
	MP	OUTPUT+3(5),FACTOR	OUTPUT+3 xFACTOR
	UNPK	OUTPUT+3(9),OUTPUT+3(5)	UNPACK RESULT
	BC	15,*	

PACK, CP

The length attributes of the operands are adequate: the L fields can be omitted.

ZAP

OUTPUT + 3 is a relative address: of the first byte following the output part number. Because it is used often, it could be defined as OUT, for example, as follows:

NAME	OP	OPERAND
OUT	EQU	OUTPUT+3

The instruction operand would then be OUT(4),QTY.

Note that the symbols in the expression equated must all have been previously defined.

BC

*, the location counter is the address of the BC instruction.

Exercise 4

Data in the format shown, is read into the defined storage areas. Z's represent zones, X's represent decimal digits, S represents a sign, and the carat (^) represents an assumed decimal point. Write the instructions to perform the specified operations. If overflow can occur during an operation, provide a branch to OVERFL.

a. ABLE plus BAKER less FOX (result in ABLE)

```
ABLE    DS   CL7   ZO ZO ZX ZX ZX ZX SX
BAKER   DS   CL4   OX XX XX XS
FOX     DS   CL4   OO OX XX XS
```

b. ABLE less BAKER plus FOX (result in FOX)

```
ABLE    DS   CL5   ZX ZX ZX ZX SX
BAKER   DS   CL5   ZO ZX ZX ZX SX
FOX     DS   CL3   XX XX XS
```

c. ABLE plus BAKER (result in ABLE)

```
ABLE    DS   CL4   XX XX XX XS
BAKER   DS   CL4   XX XX XX XS
```

d. ABLE times BAKER plus FOX (result in FOX)

```
ABLE    DS   CL5   ZO ZO ZX ZX SX
BAKER   DS   CL1   XS
FOX     DS   CL5   ZO ZX ZX ZX SX
```

e. ABLE times BAKER less FOX (result in ABLE)

```
ABLE    DS   CL5   OO OO XX XX XS
BAKER   DS   CL2   OO XS
FOX     DS   CL6   ZX ZX ZX ZX ZX SX
```

f. ABLE times BAKER times FOX (result in GEORGE)

```
ABLE    DS   CL4   ZX ZX ZX SX
BAKER   DS   CL3   ZX ZX SX
FOX     DS   CL2   OX XS
GEORGE  DS   CL9   ZX ZX ZX ZX ZX ZX ZX ZX SX
```

g. ABLE divided by BAKER (result in ABLE)

```
ABLE    DS   CL5   ZO ZO ZX ZX+ X
BAKER   DS   CL2   OX X-
```

h. ABLE divided by BAKER (quotient in FOX, remainder in GEORGE)

```
ABLE    DS   CL5   OO XX XX XX XS
BAKER   DS   CL2   XX XS
FOX     DS   CL3   XX XX XS
GEORGE  DS   CL3   XX XX XS
```

(Cont'd. on following page)

i. ABLE divided by BAKER, quotient plus FOX (result in GEORGE)

```
ABLE    DS   CL3   XX XX XS
BAKER   DS   CL3   XX XX XS
FOX     DS   CL1   XS
```

j. ABLE times BAKER compared to FOX (branch on "less than")

```
ABLE    DS   CL3   ZX ZX SX
BAKER   DS   CL2   XX XS
FOX     DS   CL4   ZX ZX ZX SX
```

k. If ABLE is less than or equal to BAKER, add ABLE to FOX otherwise multiply FOX by BAKER (result in FOX)

```
ABLE    DS   CL2   XX XS
BAKER   DS   CL2   XX XS
FOX     DS   CL5   OO OO OX XX XS
```

l. ABLE times BAKER plus FOX (result in BAKER)

```
ABLE    DS   CL5   ZO ZO ZX^ZX SX
BAKER   DS   CL2   ^ZX SX
FOX     DS   CL6   ZO ZX ZX ZX^ZX SX
```

Exercise 5

Define the storage and constants of Exercise 2 on page 14 and recode it using symbols, expressions, and implied base registers and lengths wherever possible.

DATA MANIPULATION

This section illustrates the instructions and techniques that modify and examine logical and numeric data. Bits, decimal digits, and alpha-numeric characters will be examined and altered.

BIT MANIPULATION

The rules for manipulating bits by And, Or, and Exclusive Or instructions follow.

AND

Operand 1	Operand 2	Result
1	0	0
0	1	0
1	1	1
0	0	0

OR

0	1	1
1	0	1
1	1	1
0	0	0

EXCLUSIVE OR

0	1	1
1	0	1
1	1	0
0	0	0

CONDITION CODE

	0	1	2	3
Result is	Zero	Not Zero		

Programming note: AND may be used to set bits to zero, OR may be used to set bits to one, and EXCLUSIVE OR can be used to invert bits. (Note that Exclusive Or retains bits that do not have corresponding one bits in the mask.)

ASSEMBLY SYSTEM HEXADECIMAL CONSTANTS

Hexadecimal constants are frequently used to create masks for bit manipulation. In the following example, MASK 1 becomes the address of the first byte of a six byte ($2xL_n$) constant. The constant will be OOOFOCOOOFOC (all characters shown are four-bit hexadecimal digits.

NAME	OPERATION	OPERAND	Length Attribute
MASK 1	DC	2XL3'FOC'	3

Note that hexadecimal zeros are padded to the left of the X field to make an even number of hexadecimal digits. The maximum constant length before the duplication factor is applied is 16 bytes. Hexadecimal immediate-data takes the form: X'dd', where d is a hexadecimal digit 0000-1111 or 0-9, A-F. If L_n is too small, digits on the left will be truncated.

INSTRUCTIONS

AND (RR)	NR	$(R_1)And(R_2) \rightarrow R_1$	R_1, R_2	CC
AND (RX)	N	$(R_1)And(Sx_2) \rightarrow R_1$	$R_1, D_2(X_2, B_2)$	CC
AND (SI)	NI	$(Sb_1)AndI_2 \rightarrow Sb_1$	$D_1(B_1), I_2$	CC
AND (SS)	NC	$(Sb_1)And(Sb_2) \rightarrow Sb_1$	$D_1(L, B_1), D_2(B_2)$	CC
OR (RR)	OR	$(R_1)Or(R_2) \rightarrow R_1$	R_1, R_2	CC
OR (RX)	O	$(R_1)Or(Sx_2) \rightarrow R_1$	$R_1, D_2(X_2, B_2)$	CC
OR (SI)	OI	$(Sb_1)OrI_2 \rightarrow Sb_1$	$D_1(B_1), I_2$	CC
OR (SS)	OC	$(Sb_1)Or(Sb_2) \rightarrow Sb_1$	$D_1(L, B_1), D_2(B_2)$	CC
EXCLUSIVE OR (RR)	XR	$(R_1)XOr(R_2) \rightarrow R_1$	R_1, R_2	CC
EXCLUSIVE OR (RX)	X	$(R_1)XOr(Sx_2) \rightarrow R_1$	$R_1, D_2(X_2, B_2)$	CC
EXCLUSIVE OR (SI)	XI	$(Sb_1)XOrI_2 \rightarrow Sb_1$	$D_1(B_1), I_2$	CC
EXCLUSIVE OR (SS)	XC	$(Sb_1)XOr(Sb_2) \rightarrow Sb_1$	$D_1(L, B_1), D_2(B_2)$	CC

Note that there is only one (8 bit) L field in the SS-type instructions above. Note also that words in storage-to-register operations must be aligned on word boundaries.

Illustration

1. Location ABLE : 3+ 00111100
   Instruction: NI ABLE, X'OF' 00001111

   Location ABLE after: 0+ 00001100
   CC=1, result not all zero

2. Register 5: 0110111010 ——————— 0
   Register 6: 10111010010 ——————— 0

   Instruction: OR 5, 6

   Result: Register 5: 11111110110 ——————— 0
   Register 6 unchanged
   CC=1, result not all zero

3. ABLE 1101011101011001
   BAKER 1101011100000000

   Instruction: XC ABLE(2), BAKER

   ABLE after: 0000000001011001
   BAKER unchanged
   CC=1, result not all zero

Note that if the second field is shorter than the first field in an MVO instruction, the second field will be padded with leading zeros. The MVN and MVZ replace half of each byte in the specified field without altering the other half bytes. MVN operates on the right halves of bytes and MVZ the left halves.

ABLE	12	34	56	7+
BAKER	88	88	88	88

OPERATION	OPERAND	Result to BAKER
MVC	BAKER+2(2), ABLE	88 88 12 34
MVN	BAKER+3(1), ABLE+3	88 88 12 3+
UNPK	BAKER, BAKER+2(2)	00 Z1 Z2 +3
MVI	BAKER, C'$'	58 Z1 Z2 +3

Exercises

1. Shift the decimal field ABLE left two digits, three digits. ABLE = 12 34 56 7+
2. Shift the original ABLE field right two digits; three digits.

LOGICAL DATA SHIFTING

The contents of any register or pair of registers may be shifted left or right. Every bit is shifted. Zero bits fill vacated positions and a number of bits, equal to the number shifted, are lost. These instructions are used to eliminate unwanted data and/or to select bits for processing.

Shift Left Logical (Single)	(RS)	SLL	(R_1) Shifted Left	$R_1, D_2(B_2)$
Shift Right Logical (Single)	(RS)	SRL	(R_1) Shifted Right	$R_1, D_2(B_2)$
Shift Left Double Logical	(RS)	SLDL	(R_1, R_1+1) Shifted Left	$R_1, D_2(B_2)$
Shift Right Double Logical	(RS)	SRDL	(R_1, R_1+1) Shifted Right	$R_1, D_2(B_2)$

The low-order six bits of $(B_2)+D_2$ specify the number of bit positions to shift. RN and RN+1 must be an even-odd pair of registers.

Illustration

Registers 4 and 5 contain the following bytes: PQRS and TUVW. Register 6 contains binary zeros.

DECIMAL DATA SHIFTING

Decimal data are shifted by combinations of move and bit-manipulating instructions. (An L on the right indicates that the operation is performed left to right. An R indicates right to left.)

Move (Immediate) (SI)	MVI	$I_2 \rightarrow Sb_1$	$D_1(B_1), I_2$	
Move Numeric (SS)	MVN	Sb_2 numerics $\rightarrow Sb_1$	$D_1(L, B_1), D_2(B_2)$	L
Move with Offset (SS)	MVO	Sb_2 offset $\rightarrow Sb_1$	$D_1(L_1, B_1), D_2(L_2, B_2)$	R
Move Zones (SS)	MVZ	Sb_2 zones $\rightarrow Sb_1$	$D_1(L, B_1), D_2(B_2)$	L

OPERATION	OPERAND	Register 4	Register 5
SRDL	4, 24(6)	OOOP	QRST
SLL	4, 8(6)	OOPO	QRST
SLDL	4, 16(6)	POQR	STOO

Note that the B_2 and X_2 fields may be omitted if the D_2 field is to specify the shift amount. The first operand could be 4, 24.

CHARACTER MOVEMENT

The two following instructions are used to select a byte from storage and to store a byte in storage.

Insert Character (RX)	IC	(Sx_2) byte → R_1	$R_1, D_2(X_2, B_2)$
Store Character (RX)	STC	(R_1) byte → Sx_2	$R_1, D_2(X_2, B_2)$

In each case the byte in the register is the rightmost byte.

CONDITION CODE

OPERATION	0	1	2	3
Compare Operand 1	=	<	>	
Test Selected bits	zeros (or the mask is zero)	mixed		ones

Illustrations

1. Register 3: 011010 —— 0
   Register 4: 010111 —— 1

   Instruction: CLR 3, 4

   Condition code set: 2, first operand (R_3) greater than second operand.

2. ABLE (byte) 10101111

   Instruction: TM ABLE, X'EB'
   (note: X'EB' = 11101011)

   Condition code set: 1, some selected bits are zero, some one.

LOGICAL TESTING

The following instructions are used to test or compare bits, fields, and words.

Compare Logical (RX)	CL	Compare (R_1) : (Sx_2)	$R_1, D_2(X_2, B_2)$	CC
Compare Logical (Character) SS	CLC	Compare (Sb_1) : (Sb_2)	$D_1(L, B_1), D_2(B_2)$	CC
Compare Logical Immediate (SI)	CLI	Compare (Sb_1) : I_2	$D_1(B_1), I_2$	CC
Compare Logical (Register) (RR)	CLR	Compare (R_1) : (R_2)	R_1, R_2	CC
Test under Mask (SI)	TM	Test Sb_1 bits : I_2 bits	$D_1(B_1), I_2$	CC
Branch on Condition (RR)	BCR	Go to Sr_2 if a mask bit corresponds to a Condition Code	M, R_2	

Note that only one L field is included in the CLC instruction. Also, the I_2 field of TM is a mask; one bits select corresponding Sb_1 bits to examine. If the BCR instruction contains zero for R_2, the instruction is ignored (is effectively a non-operation). The word specified by the CL instruction must be aligned on a word boundary.

Exercises

3. The bits in location PERSON have the following significance:

	ZERO	ONE
SEX (leftmost bit)	Male	Female
EDUCATION	High School	College
MILITARY SERVICE	No	Yes
MARITAL STATUS	Single	Married
U.S. CITIZENSHIP	No	Yes
AGE OVER 35	No	Yes
UNION MEMBER	No	Yes
SAVINGS PLAN	No	Yes

   a. Write the instructions necessary to branch to statement FOUND if the person represented by PERSON is a married, U.S. citizen, over 35.

   b. Branch to statement FOUND if PERSON is a single, male, college graduate, under 35.

4. LOCATIONS OP1 and OP2 are decimal numbers to be divided: OP1 by OP2. OP1 contains seven decimal digits plus the sign and OP2 contains five decimal digits plus the sign. Without dividing, determine if a division would be performed by the computer. Both fields have length attributes of one.

5. Register 2 contains all ones. Register 3 contains a four-byte field. Determine if the rightmost byte in Register 3 is the letter A.

   a. without shifting
   b. utilizing shifts of Register 3
   c. utilizing shifts of Register 2

6. Locations UNIT 1 and UNIT 2 are records with the following contents: (see top right col.)

Field	Format
Unit Number	6 alphanumeric bytes
Unit Quantity	xx xx x+
Part Number 1	8 alphanumeric bytes
Part 1 Quantity	xx xx x+
Part Number 2	8 alphanumeric bytes
Part 2 Quantity	xx xx x+

The two units combined may have two, three, or four different parts. Prepare an output record, OUTPRT, which contains a part number and quantity for each existing part. (Note that the part numbers are in sequence; that is, each first part number is logically smaller than the corresponding second part number.) Storage must be defined.

EDITING DATA

Decimal data is unpacked and edited for output by the following two instructions shown at bottom of page.

Note that only one L field is included in these instructions. It is the length of the mask pattern. Also note that the byte address is one to the right of the last symbol-filled character position unless a significance-start symbol is encountered before a nonzero digit. General-purpose register one is always implied.

CONDITION CODE

	0	1	2	3
Result is	0	⊖	+	

EDITING SYMBOLS

Symbol	Hexadecimal Code	Explanatory Symbol Used	Function
Fill	any	any	Replaces leading zeros.
Significance-Start	21	~~S~~	Stops replacement of zeros. Also acts as digit select.
Digit Select	20	~~d~~	Specifies digit position in data (replaced by fill).
Field Separator	22	/	Indicates editing of new field to begin (replaced by fill).
Edit Symbols	any	any	Inserted in result or replaced by fill.

The most common fill characters are the asterisk (*) and the blank (~~b~~).

Edit (SS)	ED	Edit (Sb_2) by (Sb_1)	$D_1(L, B_1), D_2(B_2)$	CC
Edit and Mark (SS)	EDMK	Edit; byte address → R1	$D_1(L, B_1), D_2(B_2)$	CC

OPERATION	OPERAND	Remarks/Results
LA	1, MASK+4	(R1) = MASK+4
EDMK	MASK(7), NUMBER	MASK=~~bbb~~1.23~~bbb~~, (R1) = Mask+3
BCTR	1, 0	(R1) = MASK+2
OI	0(1), C'$'	MASK=~~bb~~$1. 23~~bbb~~ Condition Code = 2

Illustrations

Field ABLE: ~~bdd~~, ~~ddS~~. ~~ddb~~CR
Field NUMBER: 0123456+

Instruction: ED ABLE(13), NUMBER

Result: ABLE = ~~bb1~~, 234. 56~~bbb~~
Condition Code: 2: the result is positive

If ABLE were: 00 00 005-

Result: ABLE = ~~bbbbbbb~~. 05~~b~~CR
Condition Code: 1: the result is negative

If the instruction executed was EDMK, Register 1 would contain ABLE+2 in the first case. In the second case, Register 1 would contain its original contents because an address is not stored if a significance-start symbol is encountered before a non-zero digit is encountered.

To utilize the EDMK instruction for currency symbol insertion, Register 1 should initially be loaded with the address of the character following the significance-start symbol. Furthermore, after executing the EDMK instruction, the address in Register 1 must be reduced by one to address the last non-significant zero digit-position. Two instructions which may be used in this procedure are the Branch on Count instruction, which subtracts a one from the contents of a register, and the Load Address instruction.

Branch on Count (RR)	BCTR	$(R_1)-1 \rightarrow R_1$ Branch to (Sr_2) if $(R_1) \neq 0$	R_1, R_2
Load Address (RX)	LA	$(X_2)+(B_2)+D_2 \rightarrow R_1$	$R_1, D_2(X_2, B_2)$

Note that (R_1) is reduced and no branching occurs if zero is specified for R_2. (Another form and other uses of the Branch on Count instruction will be illustrated in a later section.) Note also that LA does not access storage.

Example

Edit field NUMBER and insert a dollar sign in the last blank-filled position.

Field MASK: ~~bdd~~S. ~~ddb~~CR
Field NUMBER: 00123+

The reader should verify that the Or instruction performs the desired function. (~~b~~ = 01000000, $ = 01000100 or 01011011, ASCII or EBCDIC)

Exercise 7

Field NUMBER contains two numeric quantities such as:

00 12 34 5S 07 64 3S

S indicates sign. Prepare an edit mask as a constant or constants, and write the instructions to edit the numbers for printing as:

~~bbb~~123. 45S~~bbbb~~76. 43S

If a sign is plus, the S should be blank. If a sign is minus, the S should be a -. What will the condition code be if the first number is plus and the second minus?, both numbers minus?, both plus?, the first minus and the second plus?

Exercise 8

In the preceding example, use the Insert Character and Store Character instructions instead of the OI instruction.

FIXED-POINT ARITHMETIC

FIXED POINT NUMBERS

Addresses, index quantities, counters, and data are manipulated in fixed-point arithmetic. Numbers are represented in halfwords, words, and double words.

Halfword fixed-point numbers consist of a sign bit and a 15-bit integer. Full word fixed-point numbers consist of a sign bit and a 31-bit integer. A full word can represent a maximum value of 2, 147, 483, 647 or -2, 147, 483, 648. Conversions to or from decimal form produce or use 15 decimal digits plus sign (64 bits).

Negative numbers are represented in two's complement form with the sign as the leading bit. (A number is complemented by changing its binary zeros to ones, its ones to zeros, and by adding a one to the low order bit. When the sign bit is 1, the number is a complement and represents a negative value.)

The range of numbers is illustrated below.

Maximum positive	011...11
Smallest positive (+1)	00...01
Zero (always positive)	00...00
Smallest negative (-1)	11...11
Maximum negative	10...00

BASIC ARITHMETIC

The following examples illustrate binary arithmetic with five-bit operands. (The maximum positive number would be 15; the maximum negative number is 16.)

5	00101	5		00101
+3	00011	-3	00011	11101
8	01000	2		00010

OVERFLOW EXAMPLES

Overflow occurs when a sum or difference exceeds the limits. When overflow occurs the carry into the sign position differs from the carry out.

10	01010	- 9	10111
+ 7	00111	- 9	10111
17	10001 (-15)	-18	01010 (10)

ASSEMBLY SYSTEM CONSTANTS

Halfword and full-word, fixed-point constants may be defined by the DC instruction. The form of the operand is dH 'digits' or dFLn 'digits' for halfword and full word, respectively. "d" is the duplication factor and n is the explicit length in bytes before the duplication factor is applied. The constant will be aligned at an appropriate storage boundary unless an L field is used (i.e., at an address ending in one, two, or three zeros for halfwords, full words, or double words). If the constant written is larger than the implied length (two for H and four for F) or larger than the explicit length, a constant of zero will be generated. The maximum value of a fixed-point constant depends on the number of bytes in the constant, as illustrated in the following table.

Length in Bytes	Maximum Positive	Maximum Negative
4	2, 147, 483, 647	-2, 147, 483, 648
3	8, 388, 607	-8, 388, 688
2	32, 767	-32, 768
1	127	-128

Illustrations

OPERATION	Constant OPERAND	Number of Bytes	Result	Length
			2 bytes 2 bytes	Attribute
DC	2H'-123'	4	-123-123	2
DC	H'+65000'	2	0	2
DC	F'100000'	4	100000	4
DC	FL3'100000'	3	100000	3

ASSEMBLY STORAGE DEFINITION

Storage areas for fixed-point data are defined by the DS instruction with operand fields of dH, dF, or dD for halfword, full-word, or double-word storage, respectively.

Illustrations

OPERATION	OPERAND	Bytes Reserved	Length Attribute
DS	H	2	2
DS	F	4	4
DS	2H	4	2
DS	3F	12	4
DS	D	8	8
DS	2D	16	8
DS	0D	0	

The last instruction sets the location counter at the next double-word boundary and will not reserve any storage.

INSTRUCTIONS

	Add (Register) (RR)	AR	$(R_1) + (R_2) \rightarrow R_1$	R_1, R_2	CC
	Add (Indexed) (RX)	A	$(R_1) + (Sx_2) \rightarrow R_1$	$R_1, D_2(X_2, B_2)$	CC
[1]	Add Halfword (RX)	AH	$(R_1) + (Sx_2) \rightarrow R_1$	$R_1, D_2(X_2, B_2)$	CC
	Add Logical (RR)	ALR	$(R_1) + (R_2) \rightarrow R_1$	R_1, R_2	CC
	Add Logical (RX)	AL	$(R_1) + (Sx_2) \rightarrow R_1$	$R_1, D_2(X_2, B_2)$	CC
[2]	Divide (Register) (RR)	DR	$(R_1) \div (R_2) \rightarrow R_1 + 1$	R_1, R_2	
[2]	Divide (Indexed) (RX)	D	$(R_1) \div (Sx_2) \rightarrow R_1 + 1$	$R_1, D_2(X_2, B_2)$	
	Subtract (Register) (RR)	SR	$(R_1) - (R_2) \rightarrow R_1$	R_1, R_2	CC
	Subtract (Indexed) (RX)	S	$(R_1) - (Sx_2) \rightarrow R_1$	$R_1, D_2(X_2, B_2)$	CC
[1]	Subtract Halfword (RX)	SH	$(R_1) - (Sx_2) \rightarrow R_1$	$R_1, D_2(X_2, B_2)$	CC
	Subtract Logical (RR)	SLR	$(R_1) - (R_2) \rightarrow R_1$	R_1, R_2	CC
	Subtract Logical (RX)	SL	$(R_1) - (Sx_2) \rightarrow R_1$	$R_1, D_2(X_2, B_2)$	CC
[3]	Multiply (Register) (RR)	MR	$(R_1+1) \times (R_2) \rightarrow R_1, R_1+1$	R_1, R_2	
[3]	Multiply (Indexed) (RX)	M	$(R_1+1) \times (Sx_2) \rightarrow R_1, R_1+1$	$R_1, D_2(X_2, B_2)$	
[1, 10]	Multiply Halfword (RX)	MH	$(R_1) \times (Sx_2) \rightarrow R_1$	$R_1, D_2(X_2, B_2)$	
	Compare (Register) (RR)	CR	$(R_1) : (R_2)$	R_1, R_2	CC
	Compare (Indexed) (RX)	C	$(R_1) : (Sx_2)$	$R_1, D_2(X_2, B_2)$	CC
[1]	Compare Halfword (RX)	CH	$(R_1) : (Sx_2)$	$R_1, D_2(X_2, B_2)$	CC
[8]	Convert to Binary (RX)	CVB	Convert $(Sx_2) \rightarrow R_1$	$R_1, D_2(X_2, B_2)$	
[4, 8]	Convert to Decimal (RX)	CVD	Convert $(R_1) \rightarrow Sx_2$	$R_1, D_2(X_2, B_2)$	
	Load (Register) (RR)	LR	$(R_2) \rightarrow R_1$	R_1, R_2	
	Load (Indexed) (RX)	L	$(Sx_2) \rightarrow R_1$	$R_1, D_2(X_2, B_2)$	
[1]	Load Halfword (RX)	LH	$(Sx_2) \rightarrow R_1$	$R_1, D_2(X_2, B_2)$	
	Load and Test (RR)	LTR	Test $(R_2) \rightarrow R_1$	R_1, R_2	CC
	Load Complement (RR)	LCR	$-(R_2) \rightarrow R_1$	R_1, R_2	CC
	Load Positive (RR)	LPR	$\lvert R_2 \rvert \rightarrow R_1$	R_1, R_2	CC
	Load Negative (RR)	LNR	$-\lvert R_2 \rvert \rightarrow R_1$	R_1, R_2	CC
[5]	Load Multiple (RS)	LM	$(Sb_2) \rightarrow R_1 \ldots R_3$	$R_1, R_3, D_2(B_2)$	
[4]	Store (Indexed) (RX)	ST	$(R_1) \rightarrow Sx_2$	$R_1, D_2(X_2, B_2)$	
[1, 4]	Store Halfword (RX)	STH	$(R_1) \rightarrow Sx_2$	$R_1, D_2(X_2, B_2)$	
[4, 5]	Store Multiple (RS)	STM	$(R_1) \ldots (R_3) \rightarrow Sb_2$	$R_1, D_2(X_2, B_2)$	
[6, 9]	Shift Left Single (RS)	SLA	(R_1) shifted left	$R_1, D_2(B_2)$	CC
[6, 9]	Shift Right Single (RS)	SRA	(R_1) shifted right	$R_1, D_2(B_2)$	CC
[6, 7, 9]	Shift Left Double (RS)	SLDA	(R_1), (R_1+1) shifted left	$R_1, D_2(B_2)$	CC
[6, 7, 9]	Shift Right Double (RS)	SRDA	(R_1), (R_1+1) shifted right	$R_1, D_2(B_2)$	CC

Notes

1. Halfwords are fetched from memory and the sign is propagated to the left to produce a full word before the operation is performed. Halfwords are stored from the right half of the specified register; the left half is ignored.

2. The dividend must be a 64-bit number located in an even-odd pair of registers. R_1 must specify the even register. The quotient appears in the odd register. The remainder appears in the even register; its sign is the sign of (R_1). The divisor (Sx_2) or (R_2) must be larger in absolute value than the first word of the dividend (R_1).

3. The product of full-word multiplication is a 64-bit number which appears in an even-odd pair of registers. R_1 must be even. The multiplicand is selected from the odd register.

4. This is the only fixed-point instruction in which the result appears in the operand 2 location.

5. The multiple operations use consecutive registers (R_1 to R_3) and consecutive words in storage. Register 0 follows Register 15.

6. Right-shift operations propagate the sign. Overflow results during left-shift operations if a bit different from the sign is shifted out of position 1 (the position to the right of the sign). The sign bit does not change when a shift is performed.

7. Double shifts utilize an even-odd pair of registers and R_1 must be even.

8. In binary/decimal conversions, the decimal operand occupies a double word and must be aligned at an integral storage boundary.

9. Shifting a number is equivalent to multiplying (shift left) or dividing (shift right) by a power of two.

10. Multiply Halfword yields a 32-bit product.

All operands and results in storage must be located on integral storage boundaries.

Programming notes

The complement of zero is zero. When the same register is specified for R_1 and R_2 in subtraction, the register is cleared to zero.

Because of the settings of the condition code, a left shift of zero places can be used as a sign and magnitude test.

CONDITION CODE

OPERATION		0	1	2	3
Load Positive		0		+	overflow
Load Negative	R	0	⊖		
Add, Subtract	E	0	⊖	+	overflow
Shift Left, Load Complement	S	0	⊖	+	overflow
* Add Logical	U	0, NC	≠0, NC	0, C	≠0, C
* Subtract Logical	L		≠0, NC	0, C	≠0, C
Load and Test, Shift Right	T	0	⊖	+	
Compare: Operand 1		=	<	>	

Note that multiplication and division do not set the condition code.

*C indicates that there was a carry out of the sign position. NC indicates that there was no carry.

Illustrations

1. FIELD ABLE (32 bits): 0 ——010101 = + 21
   FIELD ABLE+4 (32 bits): 0 —— 01110 = + 14
   FIELD ABLE+8 (32 bits): 0 —— 10110 = - 10

OPERATION	OPERAND	Result
LM	3, 5, ABLE	(R3) = 21, (R4) = 14, (R5) = -10
AR	3, 5	(R3) = 11: 0—01011
CR	4, 3	CC = 2, (R4) > (R3)

2. Two eight-digit decimal numbers in zoned format are located at fields N1 and N2. N1 is to be divided by N2 in fixed-point arithmetic. Assume that N1 is +75 and N2 is +4.

NAME	OPERATION	OPERAND	Results/Remarks
N1	DS	CL8	Define 8 byte storage
N2	DS	CL8	Define 8 byte storage
NUMBR1	DS	D	Define double word
NUMBR2	DS	D	Define double word
	PACK	NUMBR1, N1	NUMBR1= 00000000000075+
	PACK	NUMBR2, N2	NUMBR2= 00000000000004+
	CVB	3, NUMBR1	(R3)= 0 ————01001011
	CVB	4, NUMBR2	(R4)= 0 ———— 00000100
	SR	2, 2	(R2)= 0 ———————— 0
	DR	2, 4	(R3)= 0 ———— 010010(18)
			(R2)= 0 ———— 000011(3)

3. The one-byte field PERSON contains single-bit codes in the right three bit positions representing sex, marital status, and citizenship as follows:

```
0 0 0 0 0 S M C
          M S Alien  0 } binary
          F M U.S.   1 } value
```

Thus, 00000101, indicates a female, single, U.S. citizen. Field FACTOR is a binary four: 00000100.

The following coding causes a branch in the instructions which process each individual case. (Note that there are eight possibilities, each represented by a binary number: 00000000 - 00000111.) Assume that the assembled coding is loaded starting at location 1000.

Storage Location	NAME	OPERATION	OPERAND	Results/Remarks
1000		SR	3, 3	Clear Register 3
1002		SR	4, 4	Clear Register 4
1004		IC	3, PERSON	(PERSON) R3: Multiplicand
1108		IC	4, FACTOR	4 R4: Multiplier
1112		MR	2, 4	(PERSON) x (FACTOR) R2, R3
1116		BC	15, FIRST (3)	Branch to FIRST + product
1120	FIRST	BC	15, MSA	Code 000 x 4 = FIRST
1124		BC	15, MSU	Code 001 x 4 = FIRST + 4
1128		BC	15, MMA	Code 010 x 4 = FIRST + 8
1132		BC	15, MMU	Code 011 x 4 = FIRST + 12
1136		BC	15, FSA	Code 100 x 4 = FIRST + 16
1140		BC	15, FSU	Code 101 x 4 = FIRST + 20
1144		BC	15, FMA	Code 110 x 4 = FIRST + 24
1148		BC	15, FMU	Code 111 x 4 = FIRST + 28
1152	MSA	.	.	Process Male Single Alien
		.	.	
		.	.	
	MSU	.	.	Process Male Single U.S. citizen
		.	.	
		.	.	
		.	.	

Note that the manipulation of PERSON could have been performed by the following sequence of instructions:

```
SR      3,3
IC      3,PERSON
AR      3,3
AR      3,3
```

This sequence of instructions uses one register instead of two, and four instructions instead of five, as in the example.

Note that the significant bits of the product will be right aligned in register three. Because the branch instruction at location 1116 is an indexed instruction, the product (R3) is added to FIRST to create the effective branch address. The factor four is used because there are four bytes in each branch instruction in the sequence of instructions beginning with FIRST. Thus, if PERSON contained 00000101 (5=FSU), the product would be 0-010100(20), the effective branch address of the branch instruction at 1116 would be FIRST+20 = 1120+20 = 1140, and the program would branch to FSU.

Exercise 1

Fields A, B, C, and D are to be defined as full-word binary fields. Create an address for a branch instruction by adding

$$\frac{C(A+B)}{D}$$

to the address LIST.

Exercise 2

Define the following as halfwords:

A: 216
B: 34
C: 252

Perform the calculation and store result of:

(A+B) - C

a. what is the condition code?
b. write the binary result.

Exercise 3

Define the following as full words:

A: 25
B: 11

multiply the two numbers and store the result.

Exercise 4

Define the following as character constants:

A: 136
B: 29

In fixed-point arithmetic, divide A by B and store the quotient. If there is a non-zero remainder, branch to REMDR and store the remainder.

Questions

1. What is the condition code for an invalid divide operation? multiply operation?

2. How will the results of a multiply operation be effected under the following circumstances:

   a. the multiplicand is in register 10?
   b. the instruction specified register 11 as the first operand?

3. List the rules for correct multiplication and division.

4. What is the effect on the value of a halfword when it is extended to a full word during a halfword arithmetic operation.

PROGRAMMING TECHNIQUES

This section defines and illustrates the machine and assembly instructions used for address manipulation, data translation, subroutine linkage, program switch operations, and program looping.

ADDRESS MANIPULATION

Much data processing involves the calculation and manipulation of addresses. One illustration of this was included in the previous section: the calculation of a branch address to select a particular processing routine. This section describes a useful constant form and the creation of the B and D fields of instructions.

EXPRESSION CONSTANT

Addresses or other expressions can be defined with the A constant as illustrated below:

NAME	OPERATION	OPERAND
ABLE	DC	AL2(A-B)

The halfword at ABLE contains the address or value of A-B. A-B may be absolute or relocatable. It may be relocatable only if the length is implied (four) or if Ln is three or four. It may be negative if it is absolute. Note that Ln must be less than or equal to four and if included will prevent boundary alignment of the constant. If L is not present a full word, appropriately aligned, will be produced.

Illustration

The following instructions load the address of ALPHA in register six.

Storage Location	NAME	OPERATION	OPERAND	Result
1000	ALPHA	DS	4C	1000-1003 reserved
.		.	.	
.		.	.	
2000	ALADRS	DC	A(ALPHA)	(2000-2003) = 1000
.		.	.	
.		.	.	
2700		L	6,ALADRS	(R6) = 1000

Note that L 6, ALPHA would load the contents of (1000-1003) in register 6. (Note also that LA 6, ALPHA would have the same result as L 6, ALADRS.)

USING, DROP INSTRUCTIONS

The USING instruction informs the assembly system that a specific register contains a particular address. The assembly system uses the registers so defined to create displacement and base register fields for instructions with implied base registers and lengths. The DROP instruction informs the assembly system that a register is no longer available for displacement and base register field creation. A program may alternately assign and drop registers as required throughout a program.

The format of the two instructions are:

NAME	OPERATION	OPERAND
	USING	Relocatable expression, simple absolute expression
	DROP	Simple absolute expression

The simple absolute expressions are normally register numbers.

The following instructions load register 3 and inform the assembly system that it contains ALPHA.

```
LA      3,ALPHA
USING   ALPHA,3
```

BRANCH AND LINK INSTRUCTION

The Branch and Link machine instruction, BALR(RR) loads a register to be used in conjunction with USING. R_2 is specified as zero to avoid a branch. When BALR is executed, the address of the instruction following BALR is loaded in R_1. (Other uses of Branch and Link instructions will be illustrated later in this section.)

The coding on the following page illustrates the USING and BALR instructions. The USING instructions inform the assembly system that registers 4, 5, 6, and 7 contain addresses 0002, 4097, 8192, and 9000, respectively. Note that the LA instruction loads the address specified by its operand field whereas the L instruction loads a storage field, which is an address.

Note also that because R6 contains 8192 and R7 contains 9000, either register could theoretically address ALPHA (9000) or ALPHA+6 (9006). R7 is used, however, because it produces the smaller displacements. If two registers would produce equal displacements, the higher numbered register would be used. (Additional examples of USING and address manipulation are given later in the section.)

DATA TRANSLATION

The Translate instruction translates characters from one form to another, such as ASCII code to EBCDIC. Similarly, fields are tested for the presence or absence of certain characters, by the Translate and Test instruction. The latter procedure is usually repetitive and may be simplified by using the Execute instruction.

CONDITION CODE (TRT)

0 all Sb_2 bytes are zero.
1 a non-zero Sb_2 byte was found.
2 the last Sb_2 byte was non-zero.

Illustration

The following illustrates the nature of the Translate instruction.

Field LIST:	$C_1, C_2, C_3, \ldots, C_L$	L characters to translate
Field TABLE:	$E_1, E_2, E_3, \ldots$	Table of up to 256 elements to translate C's
Instruction:	TR LIST (L), TABLE	

Each character (Cn) is replaced as follows:

(TABLE + Cn)→ LIST + n - 1 n = 1, 2, 3, . . . , L

Note that (TABLE + Cn) is the element E_n

If characters were to be translated from ASCII code to EBCDIC code, the decimal digit portion of the table would appear as follows, assuming that the code for zero is in location 3080 (Because 3000 is the address of the first location of the table, 3080 is the location selected by the character whose binary configuration equals 80. That character is ASCII zero: 01010000.)

Translate (SS)	TR	Select Sb_1 bytes. Corresponding Sb_2 bytes→ Sb_1	$D_1(L, B_1), D_2(B_2)$	L
Translate and Test (SS)	TRT	Select Sb_1 byte. If corresponding Sb_2 byte ≠0, Sb_2 byte →R2, Sb_1 byte address→ R1. If Sb_2 byte = 0, select next Sb_1 byte.	$D_1(L, B_1), D_2(B_2)$	CC, L
Execute (RX)	EX	Execute instruction at Sx_2 after (R_1), bits 24-31, are Or'd with (Sx_2), bits 8-15.	$R_1, D_2(X_2, B_2)$	CC*

*Condition Code setting depends on the instruction being executed. The Or operation does not alter storage or register contents. Note that bits 8-15 are the L, R, or I portions of an instruction.

Location:	3000	...	3080	3081	3082	...	3089
Contents:		...	11110000	11110001	11110010	...	11111001
EBCDIC			0	1	2		9

Storage Location	NAME	OPERATION	OPERAND	COMMENTS
0000		BALR	4,0	0002 → R4
		USING	*,4	
0002	BEGIN	LA	5,*+4095	4097 → R5
		USING	BEGIN+4095,5	
0006		LA	6,4095(0,5)	4095+4097 → R6
		USING	BEGIN+8190,6	
0010		L	7,ADALPH	9000 → R7 — operand becomes
		USING	ALPHA,7	7,1118(0,4)
		.		
		.		
1120	ADALPH	DC	A(ALPHA)	4 BYTES:0—09000
		.		
		.		
2740		AP	ALPHA(6),ALPHA+6(6)	operand:0(6,7),6(6,7)
		.		
		.		
6430		BC	15,BEGIN	operand:15,0(0,4)
		.		
		.		
9000	ALPHA	DS	80C	

Thus, if an ASCII 2 ($01010010_2 = 82_{10}$) were added to 3000, the result (3082) would be the address of EBCDIC 2. The latter value replaces the corresponding byte in the Sb_1 field.

Note that the table need contain only as many entries as there are different characters in the Sb_1 field. Therefore, although 256 locations are needed for a full table, those locations that will not be used for translation may be used for other purposes.

The Translate and Test instruction differs from Translate in the following ways: if the table entry found is binary zero, the next character in the first operand field is processed; the first operand field is never changed; when a non-binary zero table entry is found, the address of the processed character from the first operand field is loaded into register one and the table byte is loaded into register two (bits 24-31).

Illustration

Assume that INPUT is a 200-byte data string. It is composed of variable length fields. A blank separates one field from another. TABLE is a 256-byte translate table that contains binary zeros in all positions except the one corresponding to the blank (i.e., TABLE + 0100000). That location contains an A. Assume that a blank exists in location INPUT + 25.

The following instructions will search the INPUT string. (The L field of the TRT instruction is zero and is Or'd with (R3). The effective L is, therefore, 200.)

NAME	OPERAND	OPERATION	1	2	3
LENGTH	DC	F'199'	-	-	-
INPUT	DS	200C	-	-	-
TABLE	DS	256C	-	-	-
	LA	1,INPUT	INPUT	-	-
	L	3,LENGTH	INPUT	-	199
	EX	3,TRANS	INPUT+25	A	199
	.	.			
	.	.			
	.	.			
TRANS	TRT	0 (0,1), TABLE	INPUT+25	A	199
		D_1 L B_1			

Exercise 1

Write the instructions that will move the field bound by the blank found by the illustrative coding. Also, write the additional instructions necessary to enable the EX and TRT instructions to find each subsequent blank. (The instructions in the illustration may require modification.) Hint: execute the move instruction with an Execute instruction.

SUBROUTINES

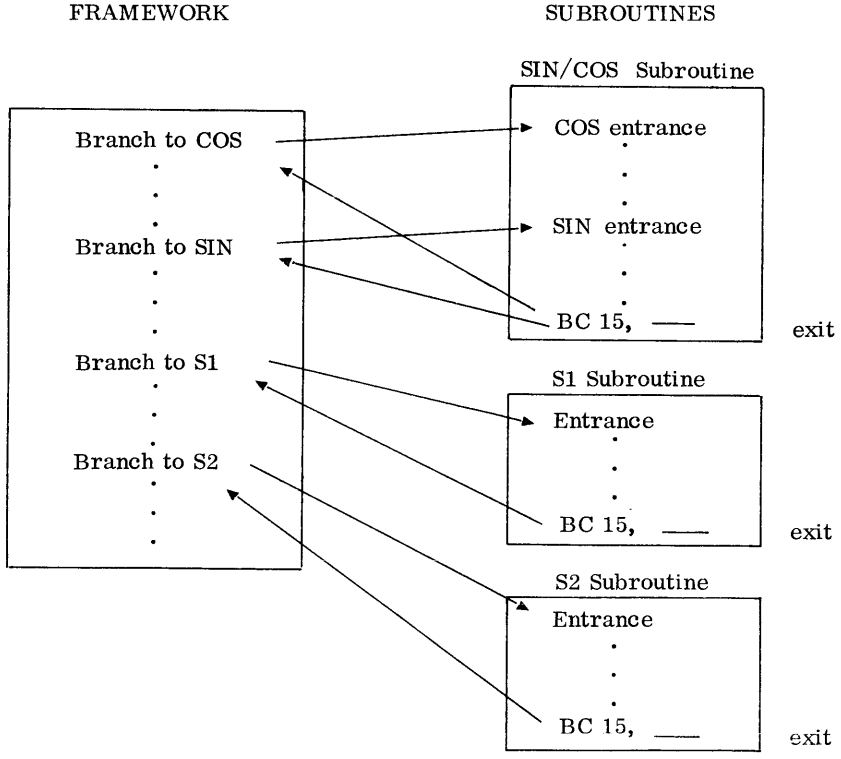
A convenient method of writing a program is to prepare a framework section to direct the logic of the program. The framework performs some of the total processing but most can be performed by branching to subroutines. The subroutines will perform well-defined portions of the job. A subroutine may have several entrances, which are locations to branch to, and usually one exit: a branch instruction to return to the framework. The following diagram illustrates this process. Note that the exit branch addresses must be modified to enable them to return to the proper location in the framework.

P-COUNTER

One of the computer operations performed for every instruction is the generation of the address of the next sequential instruction. This address is stored in the P-Counter: a word in the scratchpad memory. If a branch is performed, the branch address replaces the address in the P-Counter. When a BAL or BALR instruction is performed, the address in the P-Counter is transferred to R_1, before the branch address is placed in the P-Counter.

The address is called a link address because it is used by the subroutine as a link with the framework. Specifically, the subroutine uses the link address to create a branch address for the exit branch instruction.

Thus, the framework branches to a subroutine via a BAL instruction. The subroutine stores or retains the link address. After completing the subroutine, the exit branch instruction uses the link address to return to the framework.



Subroutine programming is facilitated by the Branch and Link instructions.

Branch and Link (RR)	BALR	Go to Sr_2 or Sx_2	R_1, R_2
Branch and Link (RX)	BAL	Link address → R_1	$R_1, D_2(X_2, B_2)$

Illustration

The following instructions illustrate the use of subroutines and the BAL and BALR instructions.

Storage Location	NAME	OPERATION	OPERAND	Remarks/Results
1000	ABLE	DS	F	
		.	.	
		.	.	
1990		LA	4, SUB	4064→R4
1994		BALR	3, 4	Go to 4064; 1996→R3
1996		.		
		.		
		.		
3046		BAL	3, COS	Go to 5000; 3050→R3
3050		.		
		.		
		.		
4064	SUB	Subroutine processing		Do not alter (R3)
		.		
		.		
	EXIT 1	BCR	15, 3	Go to 1996 = (R3)
		.		
		.		
5000	COS	ST	3, ABLE	(R3) = 3050→ABLE
		Subroutine processing		
		.		
		.		
		L	3, ABLE	(ABLE) = 3050→R3
	EXIT 2	BCR	15, 3	Go to 3050

The address 1996 and 3050 are the addresses of the instructions following the branch instructions at 1994 and 3046, respectively. Subroutine SUB must not alter (R3) because its exit branch utilizes (R3). Subroutine COS stores (R3) so that R3 may be used by the subroutine.

Exercise 2

Write the instructions that perform the following functions in fixed-point arithmetic:

1. Go to a subroutine to calculate x.
2. Multiply x by two after returning from the subroutine.
3. Go to a subroutine to add the product to a sum.
4. Go to the calculate-x subroutine.
5. Multiply x by four after returning.
6. Go to the add subroutine.
7. Go to step 1.

Ignore the actual calculation of x and the selection of successive values of x.

PROGRAM SWITCHES

It is often necessary to perform some function, perhaps by a subroutine, and then branch to one of several locations depending on predetermined conditions. Such a branch instruction is called a program switch.

Based on the pre-determined conditions, the switch is set to branch to the appropriate location.

Illustration

The following instructions illustrate a switch at location SWA. It is set to branch to A1 or A2 depending on whether OPN1 or OPN2 is performed before subroutine CMPUTE.

Storage Location	NAME	OPERATION	OPERAND	Results/Remarks
1000	OPN1	.		Operation one is performed
		.		
1090		LA	3, A1	1600→R3
1094		BC	15, CMPUTE	Go to 1400
1096	OPN2	.		Operation two is performed
		.		
		LA	3, A2	1700→R3
		BC	15, CMPUTE	Go to 1400
		.		
		.		
1400	CMPUTE	.		Perform CMPUTE
		.		
1530	SWA	BCR	15, 3	Go to 1600 or 1700 depending on (R3)
		.		
		.		
1600	A1	.		
		.		
1700	A2	.		
		.		

OPN1 ends by setting SWA to A1 (LA 3, A1). The branch instructions at SWA will therefore branch to A1, because (R3) = A1. Note that CMPUTE must not alter (R3). If R3 is needed by CMPUTE, (R3) must be stored and then reloaded before exiting at SWA. (Program switches may also be programmed by modifying or replacing instructions, or the contents of base or index registers; and by testing bits that have been set to indicate conditions.)

PROGRAM LOOPS

The two preceding exercises are examples of program loops; that is, a function is performed, and the program loops back to some point to process the next item, record, or value. In the preceding case subsequent values of x must be selected. Four instructions that are useful in programming loops are:

Branch on Count (RR)	BCTR	$(R_1) - 1 \rightarrow R_1$	R_1, R_2
Branch on Count (RX)	BCT	Go to Sr_2 or Sx_2 if $(R_1) \neq 0$	$R_1, D_2(X_2, B_2)$
Branch on Index High (RS)	BXH	See below	$R_1, R_3, D_2(B_2)$
Branch on Index Low or Equal (RS)	BXLE	See below	$R_1, R_3, D_2(B_2)$

BXH and BXLE: $(R_1) + (R_3) \rightarrow R_1$; Compare (R_1) : (R_a)

[$R_a = R_3$ or $R_3 + 1$, whichever is odd]

BXH: after the above, go to Sb_2 if $(R_1) > (R_a)$

BXLE: after the above, go to Sb_2 if $(R_1) \leq (R_a)$

A loop using the BX instructions is comprised of index-type (RX) instructions. Loops that do not consist of index-type instructions can use the BCT instructions to count the number of executions of the loop. In a loop of index-type instructions the index registers are incremented to address successive values. In a loop of decimal instructions, however, the contents of the base register or registers have to be modified to address successive values. Caution must be exercised when modifying base registers, however, because the registers modified may have been used to address instructions or other data.

Exercise 3

Write the instructions to perform the following functions. Use a program switch.

1. Calculate x.
2. Go to step 3. (The next time through the loop go to step 6; next time to step 3, etc.)
3. Multiply x by two.
4. Go to a subroutine to add the product to a sum.
5. Go to step 1.
6. Multiply x by four.
7. Go to step 4.

Illustration

Compute the sum of the 100 packed decimal numbers beginning at location NUMBER. Each number is two bytes long. The sum may be three bytes long.

Storage Location	NAME	OPERATION	OPERAND	Results/Remarks
0000		BALR	2, 0	0002 → R2
		USING	*, 2	
0002		LA	3, NUMBER	0045 → R3
		USING	NUMBER, 3	
0006		L	4, LIMIT	99 → R4
0010		ZAP	SUM, NUMBER	First number → SUM
0016	LOOP	A	3, INCRMT	(R3)+2 → R3
0020		AP	SUM, NUMBER	(SUM)+ current number → SUM
0026		BCT	4, LOOP	(R4)-1 → R4, if ≠0, Go to LOOP
0030		BC	15, END	Go to END on completion
0034	INCRMT	DC	F'2'	Increment for addresses
0038	LIMIT	DC	F'99'	Limit on additions
0042	SUM	DS	CL3	Sum storage
0045	NUMBER	DS	100CL2	Number storage

The instruction at LOOP causes R3 to address the current number: location 0047 after its first execution, 0049 the next time, etc. The instruction AP SUM, NUMBER is equivalent to AP 40(3, 2), 0(2, 3). 40(3, 2) always addresses location 0042. 0(2, 3) will, however, address successive numbers because (R3) is incremented by two by the instruction at LOOP after every addition to the sum. The BCT instruction reduces the limit of 99 (there will be 99 additions to the sum) and branches to LOOP except after the 99th addition. At that time (R4) = zero. Note that the limit of 99 could have been loaded by the instruction LA 4, 99.

Notice that either R2 or R3 could theoretically address NUMBER. R3 would be used by the assembly system because it results in a smaller displacement.

Exercise 4

Delete the instruction LA 3, NUMBER and the following USING instruction in the illustration. Change A 3, INCRMT to A 2, INCRMT. Will the resulting program run correctly? Why or why not?

Exercise 5

Recode the preceding example without using a BCT instruction. Test (R4) against an address limit to determine when all numbers have been added.

Exercise 6

Recode Exercise 3 using the BCT instruction. The calculation of X, to be performed using decimal arithmetic, is: X=2Y+3. There are 100 values of Y, each of which is two bytes in length. Define storage for the 200 numbers and the resulting sum. (The sum will be $2X_1+4X_2+2X_3+\ldots+4X_{100}$.)

Illustration

Move the 100 consecutive full words from area A to area B.

NAME	OPERATION	OPERAND	COMMENTS
	BALR	6, 0	R6 used for
	USING	*, 6	address generation
	L	4, INCRMT	4 → R4
	SR	3, 3	0 → R3
	LA	5, 396(0, 3)	396 → R5
LOOP	L	2, A(3)	(A+(R3)) → R2
	ST	2, B(3)	(R2) → B+(R3)
	BXLE	3, 4, LOOP	(R3)+(R4) → R3; i. e. (R3)+4 → R3
			Go to LOOP until (R3)=400
	.		(R5)
	.		
A	DS	100F	
B	DS	100F	
INCRMT	DC	F'4'	

(Note that in actual practice MVC instructions would be used to move data.)

The LA instruction creates the constant 396 and loads it into R5. The L and ST instructions operate with A and B as base addresses. These addresses are indexed (incremented) by (R3), which is zero initially, four during the next execution, etc. R3 contains 396 (4x99) after the 99th word is moved and (R3) is added to (R4); BXLE branches. After the 100th word is moved, R3 becomes 400 and BXLE does not branch; the loop is completed. Note that because R4 is used as the R_3 field in the BXLE instruction, R5 is used for the comparison.

Exercise 7

Records of binary data consisting of four words each are stored starting at location INPUT. There are ten such records. Create and store one output record for each input record, beginning at OUTPUT, by performing the following claculation:

D (A+B) - C

where A is the first word of the input record, B is the second word, C is the third word, and D is the fourth word. Each output record consists of one word. Assume that the product is contained in one word.

FLOATING-POINT ARITHMETIC

INTRODUCTION

A fundamental problem in scientific applications is the maintenance of as much significance as possible while dealing with numbers of widely varying size. Furthermore, it is desirable to have an automatic scheme for keeping records on decimal point location. The floating point instruction reperatory, an optional set of instructions, supplies these needs. Short- and long-form precision operand formats, which in turn may be normalized or unnormalized, are provided.

DATA FORMAT

Floating point operands are in one of two forms:

1. Full-word short form
2. Double-word long form

Both formats use a sign bit in position 0, followed by a characteristic in bit positions 1-7. Short-precision floating point operands contain a fraction (mantissa) in bit positions 8-31, while long-precision operands have the fraction in bit positions 8-63.

0	1-7	8-31
Sign	Charac.	Fraction in short form

CONVERSION

The step-wise process to convert numbers into their internal (machine) formats is illustrated.

1. Decompose the number into a decimal integer and a decimal fraction.

$$149.25 = 149 \text{ plus } .25$$
$$79.5 = 79 \text{ plus } .5$$
$$.01 = 0 \text{ plus } .01$$

2. The decimal integer must be converted to hexadecimal.

$$149_{10} = 95_{16}$$
$$79_{10} = 4F_{16}$$
$$0_{10} = 0_{16}$$

To convert a decimal integer to hexadecimal, divide repeatedly by 16. Each time write the remainder, starting from the right, and divide the quotient by 16 to get the next digit.

$$16 \overline{)149} \quad 9 \qquad\qquad 16 \overline{)9} \quad 0$$
$$144 \qquad\qquad 0$$
$$5 \text{ 1st remainder} \qquad\qquad 9 \text{ 2nd remainder, hence, } (95)_{16}$$

3. The decimal fraction is then converted to its hexadecimal representation.

$$(.25)_{10} = (.4)_{16}$$
$$(.5)_{10} = (.8)_{16}$$

To convert a decimal fraction to hexadecimal, multiply the fraction by 16. Write down whatever appears to the left of the decimal point as the first hexadecimal digit. Multiply the fractional part of the product by 16, again, etc.

Example:

Convert $(.25)_{10}$ to hexadecimal.

.25
16
150
25
4.00 1st digit is 4

00
16
00 A result of zero indicates that the conversion process is complete. Thus the answer is $(.4)_{16}$

Example:

Convert $(.33)_{10}$ to hexadecimal.

.33
16
198
33
5.28

.28
16
168
28
4.48

.48
16
288
48
7.68 etc. Thus, our answer to 3 places is $(.547)_{16}$

4. Combine the integral and fraction parts and express as a power of 16 (exponent).

   Thus, $(149.25)_{10} = (95.4)_{16} = .954 \times 16^2$,

   $(79.0)_{10} = (4F.0)_{16} = .4F \times 16^2$.

5. The characteristic is obtained by adding 64 (decimally) to the exponent and converting to binary.

   For $.954 \times 16^2$, the characteristic
   = exponent + base = 02 + 64
   = 66 and $(66)_{10} = (100\ 0010)_2$.

6. The fraction is converted to binary and grouped hexadecimally:

   $(.954)_{16} = (.1001\ 0101\ 0100)_2$

7. The characteristic and fraction are combined and stored in short- or long-precision form.

Illustration

149.25_{10} = 149 plus .25

$(149)_{10} = (95)_{16}$

$(.25)_{10} = (.4)_{16}$

$(95.4)_{16} = .954 \times 16^2$

characteristic = 2 + 64 = $(66)_{10} = (100\ 0010)_2$

fraction = $(.954)_{16} = (.1001\ 9101\ 0100)_2$

number (short form) = 0 1000010 1001 0101 0100 0000 0000 0000

Illustration

-95.15 = - (95 plus .15)

$(95)_{10} = (5F)_{16}$

$(.15)_{10} = (.266666)_{16}$

$(95.15)_{10} = (5F.266666)_{16}$

$(5F.266666)_{16} = .5F266666 \times 16^2$

Characteristic = 2 + 64 = $(66)_{10} = (100\ 0010)_2$

Fraction = $(5F266666)_{16} = (.0101\ 1111\ 0010\ 0110\ 0110\ 0110)_2$

Number (short form) = 1 1000010 0101 1111 0010 0110 0110 0110

NORMALIZATION

A floating point number is normalized when the high-order digit (bit positions 8, 9, 10, 11) is not zero. It is unnormalized if the high-order digit contains all zeros. A floating point operation yields the greatest precision if the fractions of the operands are normalized. There are scientific applications, however, where it is desirable not to have the processor automatically normalize all results so as to control or predict the significance of the results.

Illustration

The number $(95.4321)_{16}$ is shown in normalized form, and followed by three possible representations in unnormalized form.

$(95.4321)_{16} = .954321 \times 16^2 = 0\ 1000010\ 1001\ 0101\ 0100\ 0011\ 0010\ 0001$

$= .0954321 \times 16^3 = 0\ 1000011\ 0000\ 1001\ 0101\ 0100\ 0011\ 0010$

$= .00954321 \times 16^4 = 0\ 1000100\ 0000\ 0000\ 1001\ 0101\ 0100\ 0011$

Note: 1. It is not possible to express the number, say, as 95.4×16^0, since 95.4 is not a fraction.

2. Observe how the rightmost digits of the number are dropped as the original number is shifted to the right during the unnormalization process.

Exercise 1

Express the following numbers in normalized short-precision floating point form.

1.0
-10
.0005
193.52

Exercise 2

Express the following numbers, where possible, in short-precision floating point form with a characteristic of 1000011. Which numbers are then in unnormalized form?

450
-2
4515
.1

Exercise 3

a. Write the binary configuration (in short form) for

1. zero,
2. the largest positive number, and
3. the smallest positive number.

b. What are the advantages and disadvantages of the long-precision floating point form?

c. Without knowing the fraction, given that the exponent of A is 1000011 and that the exponent of B is 1000001, can one determine which is the larger number in absolute value?

ASSEMBLY SYSTEM CONSTANTS

Floating point constants can be generated by using the DC instruction with an operand of the form:

D 'number' or
E 'number'

for long-precision and short-precision, respectively.

Illustration

Samples of the proper uses of DC, for floating point constants, are shown below:

Name	Operation	Operand	Result, Remark
	DC	E'46.2'	Aligned at word; length is 4
	DC	D'7.29'	Aligned at double word; length is 8
	DC	D'7'	No decimal point is required
	DC	D'-12345'	Identical constants
	DC	D'-123.45E+2'	
	DC	D'-.12345E+5'	
	DC	D'-7295700E-2'	
	DC	4E'3.4'	Duplication factor is allowed

Illustration

The following DC operands are invalid:

E'25.2E76'	Exponent is too large
E3L'15.2'	L notation is not allowed
D'2.5.2'	Two decimal points
E'999999.E74'	Number is out of range
D'2.5-'	Sign must precede the number

LOAD AND STORE INSTRUCTIONS

The first group of floating point instructions to be discussed are of the data-handling type.

LOAD (long)	RR	LDR	$(R_2) \rightarrow R_1$	R_1, R_2	
LOAD (long)	RX	LD	$(Sx_2) \rightarrow R_1$	$R_1, D_2 (X_2, B_2)$	
LOAD (short)	RR	LER	$(R_2) \rightarrow R_1$	R_1, R_2	
LOAD (short)	RX	LE	$(Sx_2) \rightarrow R_1$	$R_1, D_2 (X_2, B_2)$	
LOAD and TEST (long)	RR	LTDR	$(R_2) \rightarrow R_1$	R_1, R_2	CC
LOAD and TEST (short)	RR	LTER	$(R_2) \rightarrow R_1$	R_1, R_2	CC
LOAD COMPLEMENT (long)	RR	LCDR	$(-R_2) \rightarrow R_1$	R_1, R_2	CC
LOAD COMPLEMENT (short)	RR	LCER	$(-R_2) \rightarrow R_1$	R_1, R_2	CC
LOAD POSITIVE (long)	RR	LPDR	$(\lvert R_2 \rvert) \rightarrow R_1$	R_1, R_2	CC
LOAD POSITIVE (short)	RR	LPER	$(\lvert R_2 \rvert) \rightarrow R_1$	R_1, R_2	CC
LOAD NEGATIVE (long)	RR	LNDR	$(-\lvert R_2 \rvert) \rightarrow R_1$	R_1, R_2	CC
LOAD NEGATIVE (short)	RR	LNER	$(-\lvert R_2 \rvert) \rightarrow R_1$	R_1, R_2	CC
STORE (long)	RX	STD	$(R_1) \rightarrow Sx_2$	$R_1, D_2 (X_2, B_2)$	
STORE (short)	RX	STE	$(R_1) \rightarrow Sx_2$	$R_1, D_2 (X_2, B_2)$	

Floating point operands use 4 double-word registers numbered 0, 2, 4, and 6. These are <u>different</u> from the registers numbered 0,2,4,6 among the 16 general-purpose registers. Short form instructions use only the first word of these double-word registers.

Illustration

Data handling instructions for floating-point operands are shown below.

Name	Operation	Operand	Result, Remarks
A	DC	E'46.2'	Short form 46.2
B	DC	E'-7.5'	Short form of -7.5
C	DC	E'0'	Short form 0
D	DC	D'100'	Long form 100
E	DC	D'-90'	Long form -90
F	DC	D'123E+2'	Long form of 12300
WA1	DS	D	One double word reserved
WA2	DS	2F	2 Full words reserved
	LE	0, A	46.2 to register 0; second word of RO unchanged
	LDR	2,4	Garbage? to register 2
	LER	2,0	46.2 to register 2
	LD	6,D	Long form 100 to register 6
	LTDR	4,6	Long form 100 to register 4; CC = 2
	STD	4, WA1	WA1 = long form 100
	STE	4, WA2	NOTE: WA2 = short form 100
	LE	0, B	Short form -7.5 to register 0
	LCER	2, 0	Short form 7.5 to register 2
	LNER	4, 0	Short form -7.5 to register 4
	LD	6, E	Long form -90 to register 6
	LPDR	4, 6	Long form 90 to register 4
	LCDR	4, 4	Long form -90 back to register 4

Exercise 4

The double-word field A contains a long form -15.
Load -15 in register 0 in long form.
Load -15 in register 2 in short form.
Load 15 in register 4 in long form.
Load 15 in register 6 in long form.
Do not use any other instructions not yet covered in this section.

Exercise 5

Store the second word of register 0 in both words of floating-point registers 2, 4, 6. Define any working storage necessary.

Exercise 6

Put the complement of the short form floating-point number in floating-point register 6 into General-Purpose register 3.

ARITHMETIC INSTRUCTIONS

The following chart summarizes the principal arithmetic instructions in the floating-point set.

ADD Normalized (long)	RR	ADR	$(R_1) + (R_2) \rightarrow R_1$	R_1, R_2	CC
ADD Normalized (long)	RX	AD	$(R_1) + (Sx_2) \rightarrow R_1$	$R_1, D_2 (X_2, B_2)$	CC
ADD Normalized (short)	RR	AER	$(R_1) + (R_2) \rightarrow R_1$	R_1, R_2	CC
ADD Normalized (short)	RX	AE	$(R_1) + (Sx_2) \rightarrow R_1$	$R_1, D_2 (X_2, B_2)$	CC
ADD Unnormalized (long)	RR	AWR	$(R_1) + (R_2) \rightarrow R_1$	R_1, R_2	CC
ADD Unnormalized (long)	RX	AW	$(R_1) + (Sx_2) \rightarrow R_1$	$R_1, D_2 (X_2, B_2)$	CC
ADD Unnormalized (short)	RR	AUR	$(R_1) + (R_2) \rightarrow R_1$	R_1, R_2	CC
ADD Unnormalized (short)	RX	AU	$(R_1) + (Sx_2) \rightarrow R_1$	$R_1, D_2 (X_2, B_2)$	CC
SUBTRACT Normalized (long)	RR	SDR	$(R_1) - (R_2) \rightarrow R_1$	R_1, R_2	CC
SUBTRACT Normalized (long)	RX	SD	$(R_1) - (Sx_2) \rightarrow R_1$	$R_1, D_2 (X_2, B_2)$	CC
SUBTRACT Normalized (short)	RR	SER	$(R_1) - (R_2) \rightarrow R_1$	R_1, R_2	CC
SUBTRACT Normalized (short)	RX	SE	$(R_1) - (Sx_2) \rightarrow R_1$	$R_1, D_2 (X_2, B_2)$	CC
SUBTRACT Unnormalized (long)	RR	SWR	$(R_1) - (R_2) \rightarrow R_1$	R_1, R_2	CC
SUBTRACT Unnormalized (long)	RX	SW	$(R_1) - (Sx_2) \rightarrow R_1$	$R_1, D_2 (X_2, B_2)$	CC
SUBTRACT Unnormalized (short)	RR	SUR	$(R_1) - (R_2) \rightarrow R_1$	R_1, R_2	CC
SUBTRACT Unnormalized (short)	RX	SU	$(R_1) - (Sx_2) \rightarrow R_1$	$R_1, D_2 (X_2, B_2)$	CC
MULTIPLY (long)	RR	MDR	$(R_1) * (R_2) \rightarrow R_1$	R_1, R_2	
MULTIPLY (long)	RX	MD	$(R_1) * (Sx_2) \rightarrow R_1$	$R_1, D_2 (X_2, B_2)$	
MULTIPLY (short)	RR	MER	$(R_1) * (R_2) \rightarrow R_1$	R_1, R_2	
MULTIPLY (short)	RX	ME	$(R_1) * (Sx_2) \rightarrow R_1$	$R_1, D_2 (X_2, B_2)$	
DIVIDE (long)	RR	DDR	$(R_1) / (R_2) \rightarrow R_1$	R_1, R_2	
DIVIDE (long)	RX	DD	$(R_1) / (Sx_2) \rightarrow R_1$	$R_1, D_2 (X_2, B_2)$	
DIVIDE (short)	RR	DER	$(R_1) / (R_2) \rightarrow R_1$	R_1, R_2	
DIVIDE (short)	RX	DE	$(R_1) / (Sx_2) \rightarrow R_1$	$R_1, D_2 (X_2, B_2)$	

Normalization usually takes place when the intermediate result is changed to the final result. Such a process is called post-normalization. On the instructions, designated above as unnormalized, post-normalization does not take place.

Illustration

Let (R2) = $(111)_{10}$ = 0 1000010 0110 1111 0000 0000 0000 0000

(R4) = $(110)_{10}$ = 0 1000010 0110 1110 0000 0000 0000 0000

Now, SUR 2, 4
would place 0 1000010 0000 0001 0000 0000 0000 0000

While, SER 2, 4
would place in register 2. 0 1000001 0001 0000 0000 0000 0000 0000

Illustration

(The remaining illustrations in this section use short form operands and normalized instructions.)

Compute: $A = C_i + D_i$, $i = 1, 5$.

Name	Operation	Operand	Result, Remarks
C	DC	5E'1.2'	Values for C_i
D	DC	5E'1.3'	Values for D_i
A	DS	F	
ADDR	DC	F'0'	Constants for looping
	DC	F'4'	
	DC	F'16'	
	LM	1, 3, ADDR	R1 = 0, R2 = 4, R3 = 16
	SER	0, 0	R0 = 0
LOOP	AE	0, C(1)	R0 = R0 + C_i
	AE	0, D(1)	R0 = R0 + D_i
	BXLE	1, 2, LOOP	Branch
	STE	0, A	Result to A

Illustration

Set A = A * B if C = 0

A = $(C/B)^2$ if C $\neq$ 0

Name	Operation	Operand	Remark, Comments
	LE	0, C	R0 = C
	LTER	0, 0	Set CC
	BC	8, ZPATH	Go to ZPATH if C = 0
	LE	2, C	R2 = C
	DE	2, B	R2 = C/B
	MER	2, 2	R2 = $(C/B)^2$
	BC	15, END	Jump to end
ZPATH	LE	2, A	R2 = A
	ME	2, B	R2 = A * B
END	STE	2, A	Store result in A

Exercise 7

The array A, is composed of 100 short-form floating operands. Determine the average number of the array and place it in floating-point register 6.

Condition Code Summary

Operation		0	1	2	3
ADD, SUBTRACT	RESULT	0	-	+	OF
LOAD POSITIVE	RESULT	0		+	
LOAD AND TEST	RESULT	0	-	+	
LOAD COMPLEMENTS	RESULT	0	-	+	
LOAD NEGATIVE	RESULT	0	-		
COMPARE: OPERAND 1		=	<	>	

COMPARE AND HALVE INSTRUCTIONS

The remaining, miscellaneous floating-point instructions are summarized below. It is worth noting that the compare instructions make life much easier. Comparison is algebraic, taking into account the sign. Moreover, exponent inequality is not decisive for magnitude determination because fractions may have different numbers of leading zeros in the unnormalized format.

Compare (long)	RR	CDR	(R_1) : (R_2)	R_1, R_2	CC
Compare (long)	RX	CD	(R_1) : (Sx_2)	$R_1, D_2(X_2, B_2)$	CC
Compare (short)	RR	CER	(R_1) : (R_2)	R_1, R_2	CC
Compare (short)	RX	CE	(R_1) : (Sx_2)	$R_1, D_2(X_2, B_2)$	CC
Halve (long)	RR	HDR	(R_2) / $2 \rightarrow R_1$	R_1, R_2	
Halve (short)	RX	HER	(R_2) / $2 \rightarrow R_1$	R_1, R_2	

Illustration

Find the largest number in the array A_i, i = 1, 5.
Place the number in floating-point register 2.

Name	Operation	Operand	Result, Remarks
ADDR	DC	F'4'	Constants for looping
	DC	F'4'	
	DC	F'19'	
	LM	1, 3, ADDR	R1 = 4, R2 = 4, R3 = 19
	LE	2, A	A_1 to R2
LOOP	CER	2, A(1)	R2 to A_i
	BC	4, RESET	Less than, Equal; continue
TEST	BXLE	1, 2, LOOP	
	STOP		
RESET	LE	2, A(1)	Reset R2 to A_i
	BC	15, TEST	

CONDITION CODE

The condition code settings for the floating-point operands are summarized above.

Exercise 8

Find the square root y of the number a by applying the Newton-Raphson formula of:

$$y_{i+1} = y_i + \frac{1}{2}\left(\frac{a}{y_i} - y_i\right) = \frac{a + y_i^2}{2y_i}$$

or $$y_{i+1} - y_i = \frac{1}{2}\left(a/y_i - y_i\right)$$

The procedure is to make some guess y_0 at the square root of a. A corrected second approximation y_1 is computed by then applying the formula, a third approximation by applying it again, etc. Each time an approximation is computed, it is compared with the previous one. As soon as they agree to within some value (let us use .0001), the process is stopped. NOTE: To find the nth root of the number a, one may apply the formula:

$$y_{i+1} = y_i + \frac{1}{n}\left(\frac{a}{y^{n-1}} - y_i\right)$$

Exercise 9

Evaluate $y = a_0 + a_1x + a_2x^2 + \ldots + a_nx^n$

HINT: Observe that

$$y = a_0 + x\,(a_1 + x\,(a_2 + (a_3 \ldots xa_n) \ldots))$$

The evaluation can be set up as a simple loop, starting from the innermost parenthesis and working outward. This process is called NESTING.

Assume that the fixed point <u>binary</u> value of n is in general-purpose register 1 and that x is in floating-point register 2. The coefficients are stored in HSM as words as follows:

$$a_0, a_1, \ldots a_n$$

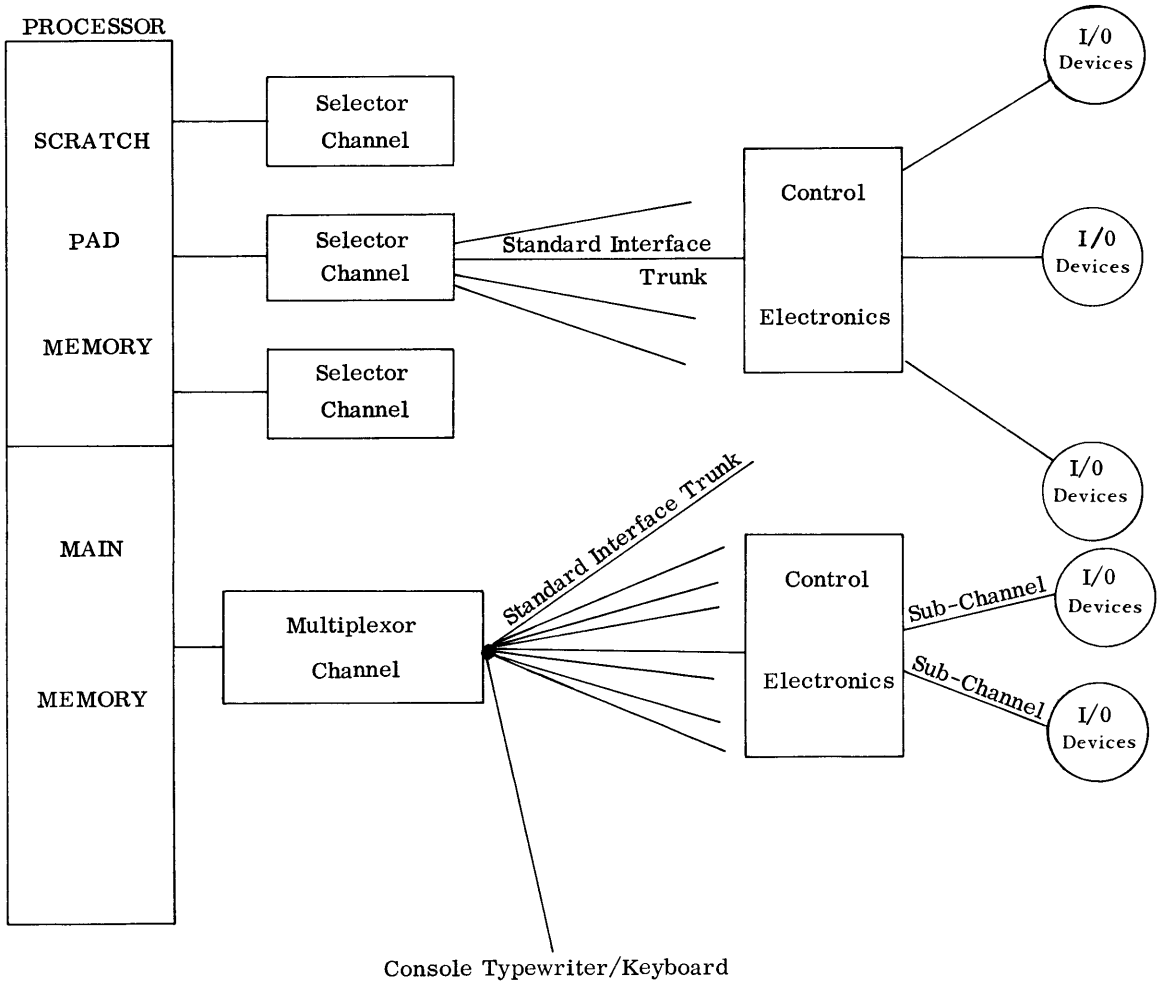
INPUT/OUTPUT

The Input/Output can be programmed with machine instructions or software instructions. This section illustrates the basic concepts of the RCA 70/45-55 systems required for either type of programming.

BASIC COMPONENTS

The following diagram shows one possible configuration of input/output components. A description of the components follows the diagram.

70/45-55 Input/Output Components



Selector Channel

Controls high-speed input/output devices. Only one device on each channel may operate at a time. An RCA 70/45 System may have three selector channels; a 70/55 may have six.

Multiplexor Channel

Controls low-speed input/output devices via eight trunks. A ninth trunk is connected to the Multiplexor channel to control a console typewriter/keyboard (optional). All devices connected to the Multiplexor channel may operate simultaneously. RCA 70/45-55 systems have one Multiplexor channel.

Standard Interface

Connects an I/O sub-system (i.e., a Control Electronics and its associated devices) to a channel.

Control Electronics

Provides immediate control over devices. Up to 16 devices may be connected to a Control Electronics. The following table illustrates the number of Control Electronics possible on each channel.

70/45 Selector - 2
70/55 Selector - 4
Multiplexor - 9

Sub/Channel

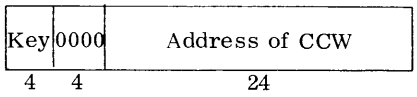
Connects an input/output device to a control electronics on a Multiplexor channel.

PRIORITY OF OPERATIONS

Channels are scanned and serviced in a priority order. Selector channel one has the highest priority and the Multiplexor channel has the lowest priority. After servicing (transferring data, etc.) is completed for one selector channel, scanning begins again with the first selector channel.

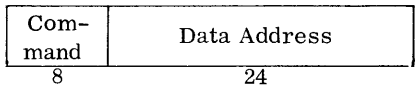
The Multiplexor trunks are also scanned in a priority order. After a Control Electronics has been serviced, scanning resumes with the first Control Electronics on the first selector channel. The Multiplexor scan is interrupted whenever a selector channel requires service. Then all selector channels are scanned, and serviced if required.

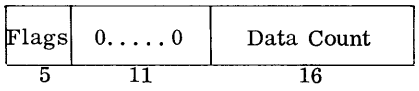
2. Channel Address Word (always Main Memory location 72)



a. Addresses a Channel Command Word
b. Specifies memory storage protection key

3. Channel Command Word





a. Specifies the input/output operation
b. Contains the address of the first byte of storage to be accessed
c. Specifies the amount of data to be transferred
d. Specifies variations to the basic operation (flags)

INPUT/OUTPUT INSTRUCTIONS

There are four input/output instructions (privileged):

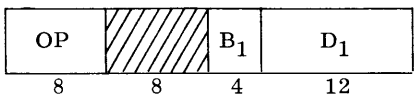
Start Device (SI)	SDV	Initiate data transmission	$D_1(B_1), I_2$	CC
Halt Device (SI)	HDV	Halt data transmission	$D_1(B_1), I_2$	CC
Check Channel (SI)	CKC	Set CC based on channel	$D_1(B_1), I_2$	CC
Test Device (SI)	TDV	Set CC based on device	$D_1(B_1), I_2$	CC

Finally, if more than one device is connected to a Control Electronics on the Multiplexor channel, the Control Electronics determines the priority of scanning its devices.

INPUT/OUTPUT OPERATIONS

Three types of control words initiate input/output operations.

1. Input/Output Instruction



a. Addresses the device
b. Initiates an operation

The CAW and CCW are used only with the Start Device Instruction. The I_2 field is not used. The rightmost eleven bits of the sum $(B_1) + D_1$ is the address of the desired device. The address has the following format.

000	Multiplexor Channel
001	Selector Channel 1
010	Selector Channel 2
011	Selector Channel 3
100	Selector Channel 4
101	Selector Channel 5
110	Invalid
111	Selector Channel 6

The four input/output instructions are used as follows:

Start Device

> initiates an input/output data transmission (if possible). The Condition Code is set to indicate whether or not the operation was initiated.

Test Device Check Channel

> The processor can test specific devices and/or channels to determine if a data transmission has been completed or if one may begin so as to optimize input/output operations.

Halt Device

> stops a data transmission so that the processor can initiate higher priority operations.

COMMANDS

The command field of the Channel Command Word (CCW) specifies the operation to be performed. The commands vary depending on the specific input/output device but certain characteristics are common to all, as shown on the following chart. The command is shown in hexadecimal. This table is illustrative only and does not include all devices. (X in the command indicates modifier bits peculiar to each device.)

BURST MODE

In the case of devices connected to a Multiplexor channel, however, one of the modifier bits indicates that the operation is performed in the burst mode. That is, the Multiplexor channel acts like a selector channel and only one device operates at one time.

COMMAND MODIFICATION

The flag bits of the Channel Command Word augment the command to:

Chain data:

> Execute a series of commands, in consecutive CCW's, such as tape write commands, to output data from non-contiguous areas of memory, with one Start Device instruction. (See Program Controlled Interrupt, which follows.)

Chain commands:

> Execute a series of similar commands, with one Start Device instruction: such as search then read from a random access device.

Suppress length indication:

> Accept as correct, an operation in which the data length does not equal the CCW count. This feature is useful, for example, when reading variable-length records.

Command	General Meaning	Magnetic Tape	Card Reader	Card Punch	Printer
X2		Read Backward			
X3	Write	Write		Punch	Print
X4		Erase			
X5	Read	Read	Read		
X7	Write Control*	Rewind	Select Stacker	Set Punch Mode	Advance Paper
X9	Transfer in channel (all): the next CCW is selected from the data address location.				
X1	Sense (all): read the sense byte (status information) into memory				

* Only one of many write control functions is shown for each device for illustrative purposes.

Skip data:

Move magnetic tape, for example, without transferring data to memory.

Program controlled interrupt:

Causes an interrupt signal when the channel control word is fetched. During chaining, CCW's, except the first one, can have this bit set. Then, after one CCW command terminates and the next CCW is fetched and initiated, an interrupt occurs informing the program that previous operation is complete.

TRANSFER IN CHANNEL

When commands or data are to be chained, the channel command words normally occupy sequential locations in memory. The Transfer in Channel command causes the selection of the next channel command word to be selected from the location specified by the data address field.

STATUS OF INPUT/OUTPUT OPERATIONS

The status of an input/output operation is determined by examining three levels of status information. These are:

1. condition code
2. scratch pad indicators (16 bits: channel status byte and standard device byte
3. sense bytes in the device

The condition code is set when any input/output instruction is issued. The condition code indicates whether or not the desired operation could be or has been performed. For many situations, the condition code contains all the information necessary for processor action. The following chart defines the Condition Code for all input/output instructions.

Condition Code	Start Device	Halt Device	Test Device	Check Channel[2]
0	Operation initiated and proceeding[2]	Operation not terminated because the Channel or Sub-channel was not busy[1, 2]	Specified device is available and can accept an operation	Channel available (Device or Sub-channel not tested)
1	Operation not initiated, consult status bits stored in Scratch Pad Memory	Operation not terminated, consult status bits stored in Scratch Pad Memory	Specified device cannot accept an operation, consult status bits stored in Scratch Pad Memory	Interruption pending in Selector Channel (Multiplexor not tested)
2	Operation not initiated because the Channel or Sub-channel was busy or an interrupt is pending[2]	Operation was terminated	Specified device cannot accept an operation because the Channel or Sub-channel is busy[2]	Selector Channel busy or Multiplexor busy operating in burst mode
3	Operation not initiated because the Channel or Sub-channel is inoperable[2]	Operation not terminated because the Channel or Sub-channel is inoperable[2]	Specified device cannot accept an operation because the Channel or Sub-channel is inoperable[2]	I/O Channel is inoperable

NOTES
1. Multiplexor Sub-channel busy but not with specified device.
2. The status bytes are not stored by this instruction or condition code.

In certain instances, the Condition Code does not convey enough information for processor action (e.g., Condition Code one for a Start Device instruction).

In such a case, the program must transfer the status bytes to main memory and then test them. (The status bytes are transferred by privileged instructions defined in a later section.) The status bytes are automatically stored in the scratch-pad memory if an operation cannot be initiated or if difficulties are encountered during the operation.

The status bytes contain information common to all devices. If they do not contain sufficient information, the sense bytes peculiar to the device must be examined by issuing a Sense Command (by a Start Device instruction) which reads the sense bytes into main memory. They are then examined to determine the action necessary. A list of status bits and an illustration of sense bytes follow.

STANDARD DEVICE BYTE (each channel)

Status modifier (during chaining a CCW is skipped due to a preceding CCW operation)
Interrupt pending (a non-input/output interrupt condition exists)
Device busy
Control busy
Device end (device operation has terminated)
Secondary indicator (sense command required)
Device inoperable
Manual Request

CHANNEL STATUS BYTE (each channel)

Program controlled interrupt (program requested interrupt, usually during chaining)
Incorrect length (data transferred $\neq$ count)
Program check (e.g., invalid data address)
Protection check (read attempted into protected memory)
Channel data check (parity error detected in channel, main or scratch-pad memory)
Channel control check (machine[channel]malfunction)
Termination interrupt pending (interrupt caused by I/O termination not effected)
Termination interrupt (I/O termination interrupt accepted).

SENSE (COMMAND) BYTE (each device)

Bit	Magnetic Tape	Card Reader	Card Punch	Printer
1		Control code sensed		Buffer available
2	Begin/End Tape			Paper low
3	Tape mark read	Manual servicing	Manual Servicing	Manual servicing
4	Tape record too small		Intervention required	Illegal character
5	Channel parity	Invalid punch	Transmission parity	Transmission parity
6	Read count error	Stacker problem	Punch buffer parity	
7	See below*	See below*		
8	Read/Write error	Read error	Punch error	

* If a data byte could not gain access to a channel or Control Electronics because of other higher-priority data transmissions, a "Service Request Not Honored" signal sets a bit in the sense byte.

TERMINATION OF INPUT/OUTPUT OPERATIONS

When an input/output operation terminates, successfully or otherwise, the program is interrupted, if the interrupt mask is set to allow interrupt. (The details of program interrupt are given in a later section.) The interrupt mask gives the program the option of accepting an interrupt when the operation terminates or at a later time.

In either case, status bytes are automatically stored in the scratch-pad memory and must be examined before another Start or Test Device instruction is issued. (A Start or Test Device instruction clears the standard device byte.)

SEQUENCE OF INPUT/OUTPUT PROGRAMMING OPERATIONS

The following is an outline of the steps required for input/output programming.

A. Issue a Start Device instruction.

B. Test the Condition Code. The Condition Code settings and the required action are shown in the following table. (See the Condition Code chart for more detail.)

CC	Indication	Action
0	Operation proceeding	Continue with program
1	Operation not initiated	Transfer status bits to memory and test
2	Operation not initiated	Issue Check Channel instruction
3	Operation not initiated	Manual action required

C. When the status bits are tested:

1. Sufficient information may be available for program action, or,

2. The sense byte(s) must be read into memory (Sense command) and tested by the program.

D. When the operation terminates and an interrupt occurs:

1. Test the status bits as in C above. (This assumes that there is an interrupt or executive routine.)

2. Take program action, if necessary, to overcome difficulties.

3. Inform the main program, by setting a program switch for example, that the operation was completed successfully.

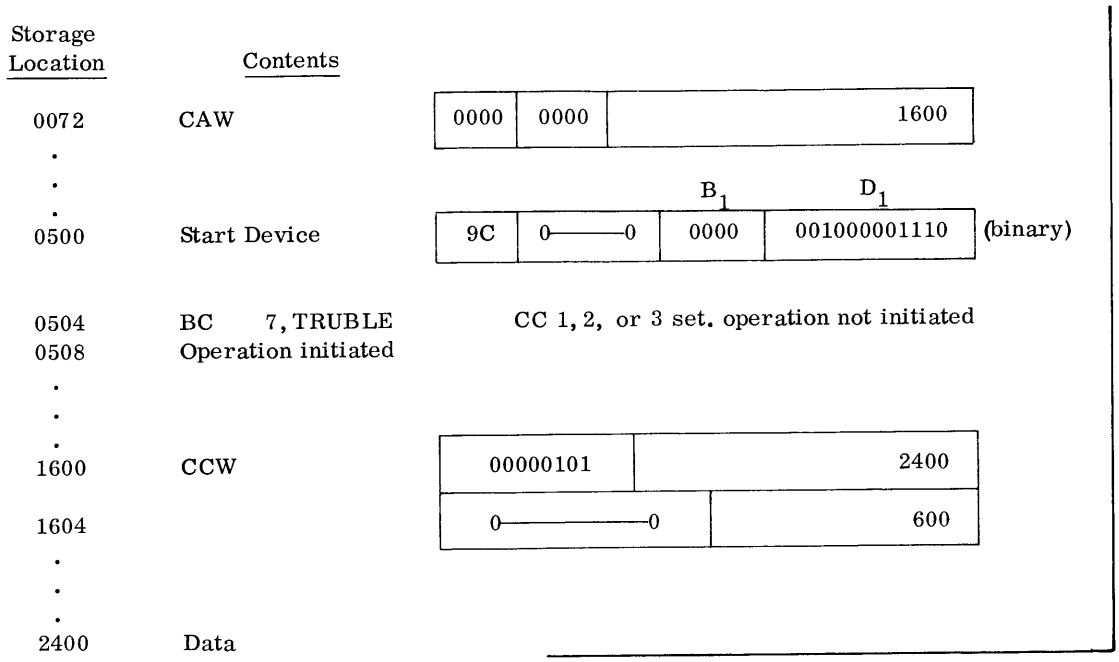
4. Return to the main program.

(If the interrupt was not accepted at termination, the program must allow the interrupt to occur before issuing another Start Device instruction. The operations in step D are then performed.)

Illustration

A record is to be read from magnetic tape:

Device number:	14
Selector channel:	2
Record length:	600 bytes
Memory data address:	2400
CCW location:	1600
No storage protection	



At location TRUBLE the exact setting of the Condition Code is determined. Status bits may require testing, sense bytes may be required, and it may be necessary to test the channel.

ASSEMBLY SYSTEM INSTRUCTIONS

Two assembly instructions that facilitate input/output programming are the Define Channel Command Word and Conditional NOP instructions.

CCW - DEFINE CHANNEL COMMAND WORD

This instruction defines an eight-byte Channel Command Word aligned on a double-word boundary. The name field is a symbol and the operand field consists of four expressions, separated by commas. They are:

1. Command Code: simple absolute expression (one byte)
2. Data address: relocatable expression
3. Flags: simple absolute expression (one byte)
4. Count: simple absolute expression (two bytes)

The following instruction creates a CCW to read 1000 data bytes into INPUT. It suppresses an incorrect-length indication (third bit of second half of CCW).

NAME	OPERATION	OPERAND
CONWRD	CCW	X'05', INPUT, X'20', 1000

Read: 00000101 Flag: 00100000

CNOP - CONDITIONAL NO OPERATION

The CNOP assembler instruction is used when it is necessary to align instructions on specific boundaries. The format of the instruction is:

NAME	OPERATION	OPERAND
Not used	CNOP	Two decimal values separated by a comma

First value: sets location counter to a specific byte in a word or double word. It may be 0, 2, 4, or 6.

Second value: specifies word (4) or double word (8).

The following are valid combinations of values.

0, 4	first byte of a word
2, 4	third byte of a word
2, 8	third byte of a double word

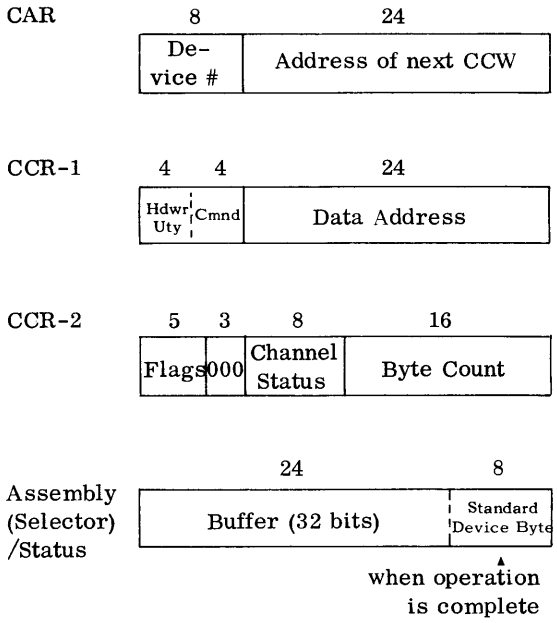
If bytes are skipped for alignment, NOP instructions (BCR) fill the skipped positions.

DATA FLOW

Data being transferred through a selector channel are controlled by addressable words in the scratch-pad memory. Furthermore, the data is read or written, one byte at a time from the assembly word in the scratch pad. This word functions as a buffer between channel and main memory. It collects four bytes before writing data or transferring input data to main memory.

Data transferred through a Multiplexor channel is read or written, one byte at a time, directly from main memory. Control words are located in main memory (not program accessible) and are transferred to the scratch pad to service the operating device as required.

The following is an illustration of the control registers. CAR is the channel address register and CCR is the channel command register.



The scratch-pad memory contains one group of these registers for each selector channel and one group for the Multiplexor. There are two additional words for each selector channel to hold the next CCW.

For each sub-channel (on a Multiplexor channel) there is one CAR, CCR-1, and CCR-2 word in main memory.

MULTIPROCESSOR CONNECTIONS

As an option, up to seven 70/45-55 processors may be connected in a multi-processor systems cluster. Each processor may signal any other and transmit a byte of information with the Read and Write Direct instructions (privileged).

Read Direct (SI)	RDD	Other processor byte → Sb_1	$D_1(B_1), I_2$
Write Direct (SI)	WRD	(Sb_1) → other processor	$D_1)B_1), I_2$

In both instructions I_2 sets up a pattern of pulses that specifies connecting lines to be sampled (read) or transmitted (write).

Questions

1. How many devices may be operating on a selector channel at one time?, on a multiplexor channel?
2. What is "burst mode"?
3. What is the function of each field of the following?
   a. Start Device instruction
   b. Channel Address Word
   c. Channel Command Word
4. Write the CCW assembly instruction that could be used to punch one full punched card from storage area CARD.
5. What is the sequence of operations that must be performed to read a block of data, initiate a second read while the program processes the first block, and then start processing the second block as soon as possible after the completion of the read.
6. What is command chaining?, data chaining? How could the latter be used advantageously during a data merging operation? When would a Transfer in Channel Command be used?
7. What is the purpose of a program-controlled interrupt during data chaining?
8. What is the difference between write and write control commands? What function does a write control command perform in a printing operation?

PROGRAM CONTROL AND LINKING

PROGRAM CONTROL INSTRUCTIONS

The following assembly instructions name programs, control the format of the program input and output, and control the storage allocation of instructions.

START

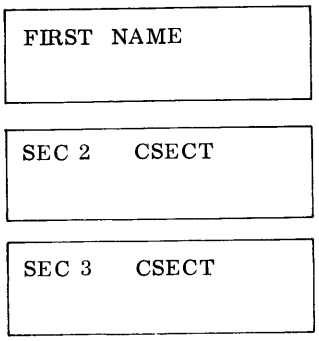
START must be the first instruction in a program. It may name the program and determine its starting location. The NAME field may be blank. If the OPERAND field is invalid or blank, the location counter is made zero. The following instruction names the program CALCLT and sets the program counter to 960.

NAME	OPERATION	OPERAND
CALCLT	START	960 (Self-defining value)

CSECT

CSECT identifies the beginning of a control section or program segment. Each section of a program that is to be loaded individually, except the first section, requires a CSECT instruction.

The following illustration represents the program FIRST. It consists of three sections, each of which is to be loaded individually.



The name field of a CSECT instruction may be blank or contain a symbol. The operand field is blank.

END

END must be the last statement in a program. If its operand field is used, it must contain a relocatable expression. This expression specifies the location of the first instruction in the program to be executed. Program MASTER, shown below, is loaded starting at location 2400 and begins with the instruction at BEGIN.

NAME	OPERATION	OPERAND
MASTER	START	2400
WORK	DS	100C
INP	DS	80C
CON	DC	A(ALPH)
BEGIN	BALR	2,0
	.	
	.	
	.	
	.	
	END	BEGIN

EJECT

EJECT causes the printer to advance the output listing so that printing continues on the next page. One use of EJECT is to separate routines in a program listing. The EJECT statement is printed prior to advancing the paper.

SPACE

The decimal operand of the SPACE instruction specifies the number of blank lines to insert in the program listing.

The following coding causes routines COS and SIN to appear on separate pages of the listing. Routine MATRIX follows routine SIN and is separated from it by five lines.

NAME	OPERATION	OPERAND
	EJECT	
COS		
.		
.		
.		
	EJECT	
SIN		
.		
.		
.		
	SPACE	5
MATRIX		

ORG

The ORG instruction resets the location counter to a new origin, or it reserves storage areas. The operand field is a relocatable expression whose value must be greater than the original value of the location counter. Symbols in the expression must have been previously defined. The following instruction causes 480 bytes to be skipped.

NAME	OPERATION	OPERAND
	ORG	* + 480

PROGRAM LINKING

Two or more programs may be assembled independently, loaded together, and then may call on one another by means of the program linking instructions, ENTRY and EXTRN. In the following illustration, program MAIN branches to or calls program SINCOS. Specifically, MAIN branches to COS, an entrance or entry point in SINCOS.

```
MAIN  START
      EXTRN    COS
        .        .
        .        .
        .        .
      BAL      15,COS
```

```
SINCOS  START
        ENTRY    COS
          .        .
          .        .
          .        .
          .        .
COS     ST       15,STOR
```

All EXTRN and ENTRY instructions must immediately follow START. There must be one EXTRN and one ENTRY for each linkage symbol (e.g. COS). There must be no more than 14 EXTRN instructions in one program. The name of a program may also be an entry point. In such a case the calling program must define the name with an EXTRN. No ENTRY instruction is required.

In addition to using ENTRY and EXTRN instructions, the calling program (e.g. MAIN) must contain an Expression Constant (e.g. A (COS)) so that the loader can store the address of the entry point. The coding at bottom of page illustrates the requirements for MAIN to branch to, and return from SINCOS.

Exercise

Two programs named ONE and TWO are to be assembled separately and loaded together. ONE has entry points A and B. TWO has entry points C and D. When ONE is assembled, the coding for A and B should each begin on separate pages of the listing. When TWO is assembled, the coding for D should begin 10 spaces after C ends on the listing.

The name of the first instruction to be executed in ONE is START, the name of the first instruction to be executed in TWO is BEGIN (if the programs were run independently). Write the control, branch, and load instructions and define the necessary constants to satisfy the above requirements and the following: TWO is to branch to ONE, A, and B; and ONE is to branch to TWO, C, and D.

```
MAIN    START
        EXTRN    COS
          .
          .
        L        4,ADCOS
        BALR     15,4
          .
          .
ADCOS   DC       A(COS)
```

This may be a relocatable compound expression such as:
COS+10

```
SINCOS  START
        ENTRY    COS
          .
          .
          .
COS       .
          .
          .
EXIT    BCR      15,4
```

This could be:

```
L       4,ADCOS
USING   COS,4
  .       .
  .       .
BAL     15,COS
```

INTERRUPT SYSTEM

CONCEPTS OF INTERRUPT SYSTEM

The interrupt system responds to asynchronously occurring external and I/O signals and monitors exceptional conditions generated by the program itself. The specifications of an interrupt system are dictated by the special needs of multi-programming and real-time processing. Some of the salient hardware that are provided in conjunction with the interrupt system are listed below:

1. Adequate Memory
2. Processing rates in excess of memory access rates.
3. Asynchronous Input/Output operations.
4. Any interrupt condition can be masked (i.e., inhibited)
5. Clock (or timer) option
6. Memory protection option
7. Minimal operator's console and operating procedures
8. Priority scheme for interrupt conditions
9. Weight register defines specific interrupt condition
10. Four processor states
11. Privileged instructions
12. Debugging option
13. Program control instruction to return to previous state
14. Read/Write Direct Control option

PROCESSOR STATES

The processor may operate in one of four states; UNIQUE registers are assigned to each state. Thus, the processor, upon interrupt, shifts states or control, rather than going through some more complex process (which probably would be partly by hardware and partly by software) of saving registers, counters, indicators, etc. and loading a new set of values from registers, counters, indicators, etc. In short, the efficiency of an interrupt system is measured in terms of time it takes to safekeep the reminders of where it was at the time of interruption, switch to the interruptive work, and then switch back to the prior work.

The four states and their associated registers are summarized at bottom of page.

In practice, the user need be concerned only with state 1. Interrupt conditions, when present and not masked, initiate either state 3 or state 4 (only machine malfunction error conditions go to state 4). Thus, the user should view RCA supplied software for states 2, 3, and 4 (such as FCP, Supervisor, interrupt analyzer, etc.) as an integral part of the system---in effect another black box.

SCAN

There are 32 possible conditions that may cause interrupt, for example, OP-code trap, decimal overflow, selector trunk 1 terminate.

Associated with these conditions are two registers:

1. Interrupt Flag Register (1 for the entire system)
2. Interrupt Mask Register (1 per processor state)

The user may permit or not permit any interrupt condition, when present, to cause interrupt by setting the appropriate bit in the Mask Register.

When an interrupt condition is present, the corresponding bit is set in the Interrupt Flag Register. This event may occur at any time. However, scanning for an interrupt condition takes place only at certain logical times:

State / Registers	1. Processing	2. Interrupt Response	3. Interrupt Control	4. Machine Condition
P Counter	1	1	1	1
General Purpose	16	16	6	5
Floating Point	4 double word	*	*	*
Interrupt Mask	1	1	1	1
Interrupt Status (ISR)	1	1	1	1
Weight	-	-	1	1

*May use floating-point registers of Processing State.

1. A flag is set in the Interrupt Flag Register.
2. The Mask Register of the current state is changed.
3. A state is initiated by the Program Control instruction.

If, say, an I/O interrupt condition is set in the Interrupt Flag Register while a floating point add operation is in progress, the add is completed before scanning for the interrupt condition takes place. With that in mind, we may summarize by saying that scanning for interrupt CAN ONLY OCCUR after the current instruction is finished. The current instruction is finished in one of three ways:

1. Instruction is completed (Normal completion)
2. Instruction is terminated (Aborted)
3. Instruction is suppressed (Never started)

P AND ISR

The P counter is composed of the following fields:

2	2	4	24
ILC	CC	Program Mask	Next Instruction Address

A. ILC, Instruction Length Code, contains the length in bytes of the last instruction executed in this state.

B. CC - Condition Code

C. Program Mask allows for a further level of masking on the following interrupt conditions:

1. Significance error
2. Exponent underflow
3. Decimal overflow
4. Fixed point overflow

D. Next Instruction Address is the address of the next instruction in sequence to be executed.

The ISR, Interrupt Status Register, is composed of the following fields:

3	5	4	1	2	1	8	8
ISI	00000	Key	A	00	N	0000 0000	Instruction bits 8-15

ISI, Interrupted State Identifier, specifies the interrupted state.

The states are designated as follows:

000	Machine Condition
001	Interrupt Control
010	Interrupt Response
011	Processing

Key is the 4-bit key used for the memory protect feature.

A specifies the internal code to be used.

A = 1 specifies ASCII while
A = 0 specifies EBCDIC.

N specifies whether or not privileged instructions may be executed by this state.

N = 0 allows them.
N = 1 does not allow them.

The rightmost 8 bits of ISR contain the R1, R2 fields of a supervisor call.

The following flow chart outlines interrupt action. Fig. 1 summarizes the effect that changing of states has on the various fields in P and ISR.

INTERRUPT (Non-program Control Type)

Figure 1

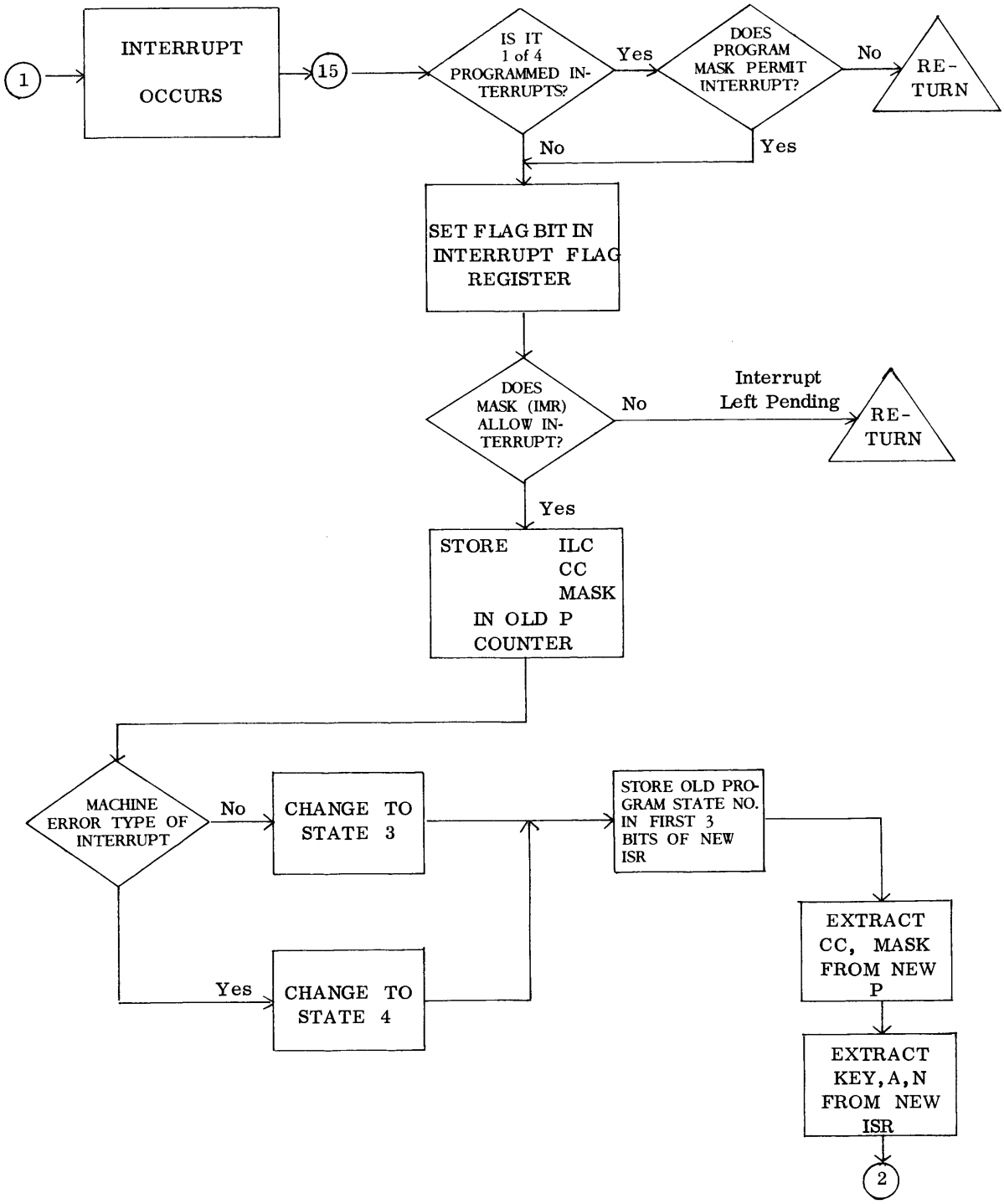
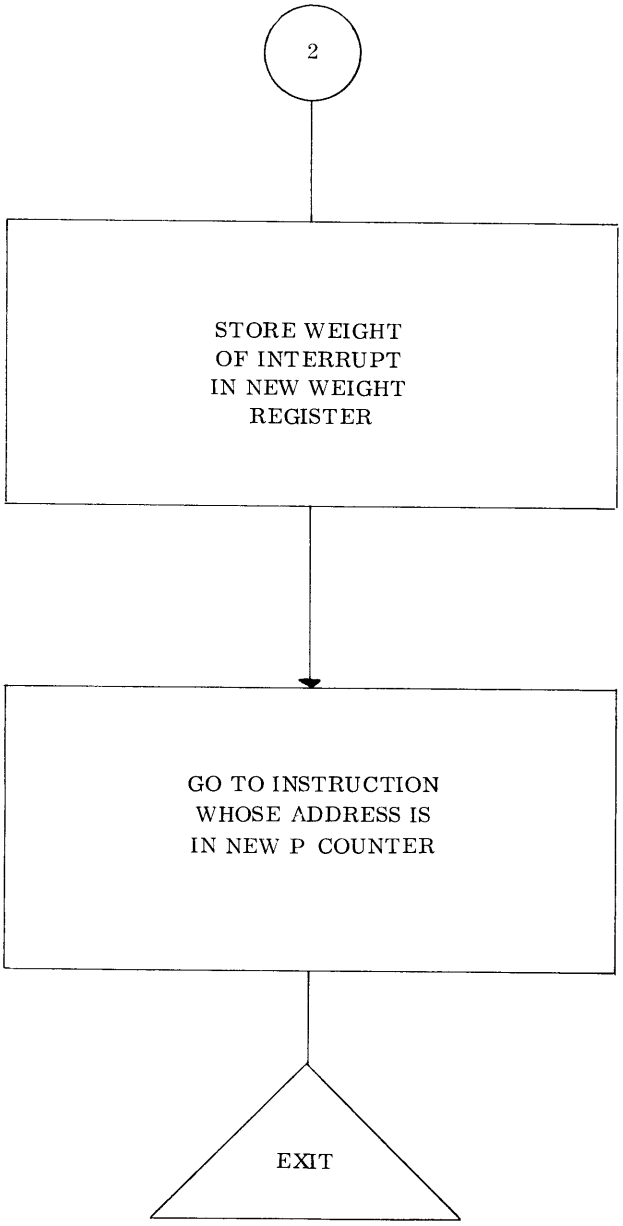


Figure 1 (Continued)



EFFECTS UPON STATES

Field	Register	Program Interrupt		Program Control (no Interrupt)		Program Control (with Interrupt)		
		Leaving	Entering	Leaving	Entering	Leaving	Entering	Designated
ILC	P-Counter	S	-	S	-	S	-	U
CC	P-Counter	S	E	S	E	S	E	U
Mask	P-Counter	S	E	S	E	S	E	U
Address	P-Counter	U	-	U**	-	U**	-	U
ISI	Interrupt Status	U	S***	U	U	U	S****	U
Key	Interrupt Status	U	E	U	E	U	E	U
A	Interrupt Status	U	E	U	E	U	E	U
N	Interrupt Status	U	E	U	E	U	E	U
R_1, R_2	Interrupt Status	U*	-	U	-	U	-	U

* Modified during Op Code Trap or Supervisor Call
** Modified by Program Control Instruction
*** Indicates state
**** Indicates state designated by Program Control
S = STORED
E = EXTRACTED (Made Machine Condition)
U = UNALTERED

INTERRUPT CONDITION SUMMARY

The principal interrupt conditions are summarized below (#1 corresponds to the high-order bit position of the Mask Register).

1. Power Failure
2. Machine Check

Go to State 4

3. External Signal No. 1
4. External Signal No. 2
5. External Signal No. 3
6. External Signal No. 4
7. External Signal No. 5
8. External Signal No. 6

Set when a signal is received on 1 of 6 external lines associated with the direct control features.

9. Not Specified
10. Selector Trunk 1 Terminate
11. Selector Trunk 2 Terminate
12. Selector Trunk 3 Terminate
13. Selector Trunk 4 Terminate
14. Selector Trunk 5 Terminate
15. Selector Trunk 6 Terminate
16. Multiplexor Trunk Terminate
17. Elapsed Time Clock : Count counts from positive to negative
18. Console Request
19. Not Specified
20. Not Specified
21. Supervisor Call : Result of execution of Supervisor Call instruction

22. Privileged Operation

23. Operation Code Trap

24. Address Error

    a. Out of memory

    b. Execute - Execute

    c. Storage and Protection Keys do not match

    d. Incorrect boundary

    e. Operand specifies odd register for a pair of double registers

    f. Floating point operations address registers other than 0, 2, 4, 6

    g. Decimal Multiplier or Divisor exceeds 15 digits and sign

    h. Block address used to set storage key has 4 low-order bits non-zero

    i. Memory protection not installed and a non-zero protection key is provided.

25. Data Error

    a. Sign or digits incorrect in decimal arithmetic, editing, or convert instructions

    b. Fields overlap incorrectly

    c. Decimal Multiplicand has too many high order significant digits

26. Exponent Overflow

27. Divide Error

28. Significance Error
29. Exponent Underflow
30. Decimal Overflow
31. Fixed Point Overflow

(28–31: May also be Controlled by Program Mask in ISR. If interrupt is not taken, the operation is completed by making the result true zero.)

32. Debugging Mode

    a. Program debugging interrupt flag is set by the Program Control Instruction

    b. When interrupt flag bit and the associated interrupt mask bit are both set, the interrupt is effected after each instruction in states 1 and 2

    c. Software can provide various trace functions

Exercise 1

Write the mask so that all Selector Trunk terminate interrupts will be inhibited.

Exercise 2

The instruction "LTER 0, 2" is being executed at location 120 in program state 1. Floating point register 2 contains a -7.5. An External Signal 3 interrupt, which is not masked out, occurs.

1. Show the contents of the old P and old ISR.
2. What state will be initiated?
3. What are the contents of the weight register of the new state?

Exercise 3

A BCTR instruction, at location 100, is being executed by an "EX" instruction at location 150 when an interrupt occurs.

What is in the old and new ILC after the interrupt takes place?

Exercise 4

A program is running in state 1 with A = 0 and N = 0 in the ISR. Describe the operation to be performed so that the program can process ASCII code.

PRIVILEGED INSTRUCTIONS

The privileged instructions, except those relating to I/O control, are summarized:

*SUPERVISOR CALL	RR	SVC	See below	R_1, R_2	
LOAD SCRATCH PAD	SS	LSP	$(Sb_2) \longrightarrow Sb_1$	$D_1(L, B_1), D_2(B_2)$	
STORE SCRATCH PAD	SS	SSP	$(Sb_1) \longrightarrow Sb_2$	$D_1(L, B_1), D_2(B_2)$	
SET STORAGE KEY	RR	SSK	See below	R_1, R_2	
*SET PROGRAM MASK	RR	SPM	See below	R_1, R_2	CC
INSERT STORAGE KEY	RR	ISK	See below	R_1, R_2	
PROGRAM CONTROL	SI	PC	See below	$D_1(B_1)$, I_2	
DIAGNOSE	SI		See below		
IDLE	SI	IDLE	See below	$D_1(B_1)$, I_2	

*Not a privileged instruction.

Supervisor Call — communicates with the operating system by transferring control to state 3. Requests and directives to the operating system are stipulated by the immediate operand R_1R_2.

Set Storage Key — specifies the key (for the memory protection option) of a designated block.

Insert Storage Key — stores (inspect) the storage key of a designated block.

Set Program Mask — specifies the 2nd level of masking for:

1. Significance error.
2. Exponent underflow.
3. Decimal overflow.
4. Fixed point overflow.

The CC is also reset by this instruction.

Program Control — terminates the current state.

Normally, RCA software (e.g. EXEC, FCP) returns control to the Processing State via this instruction.

The principal functions of Program Control are to:

1. Terminate the present state.
2. Reset P counter of state being left to D_1 (B_1).
3. Initiate new state and set controls as specified by I_2.

The 8 bits of the immediate operand I_2 are subdivided as follows: 000 D SSS R, where

D=1 Set program debugging mode for initiated state. Scan of interrupt flags is DELAYED until the first instruction of the interrupted state is executed. Then, SCAN is performed in usual way.

D=0 Debugging mode not to be effected.

SSS State desired to enter.

000 - P4
001 - P3
010 - P2
011 - P1

R=1 Specifies indirect state control (go to state specified in IS of ISR of this state). Thus, ignore SSS.

R=0 Go to state specified by SSS.

Diagnose — The instruction is for machine troubleshooting and should not be used.

Idle — The instruction effects the Idle Mode by causing the processor to continuously branch on itself. Any interrupt occurring during the Idle Mode will be taken.

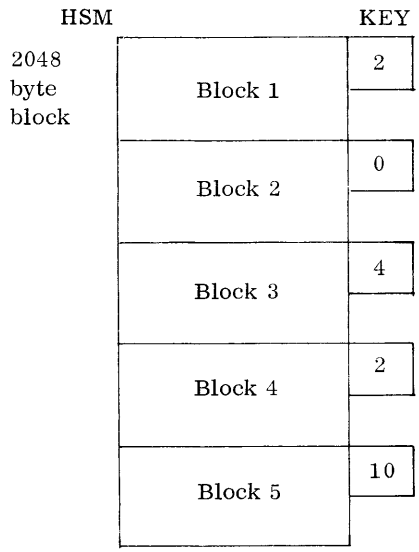
MEMORY PROTECTION

Main storage is divided into blocks of 2048 bytes. A four-bit storage key is associated with each block; the SSK instruction changes the key, while the ISK instruction stores the key.

When data is stored in a block, the storage key is compared with the protection key. If storing is speci-

fied by an instruction, the key of the ISR is used but if storing is specified by an I/O operation, the key supplied in the Channel Address Word (CAW) is used. The keys match if they are equal or when either is zero.

Illustration



Assume ISR Key = 2

STORE into Block 1	Valid
STORE into Block 2	Valid
STORE into Block 3	Interrupt
STORE into Block 4	Valid
STORE into Block 5	Interrupt
STORE into Blocks 4, 5	Interrupt

(operation terminates when crossing over from Block 4 to Block 5)

Assume CAW Key = 4

READ into Block 1	Interrupt
READ into Block 2	Valid
READ into Block 3	Valid
WRITE from Block 4	Valid
READ into Block 4	Interrupt

CLOCK

The elapsed time clock occupies the word at location 80 in main memory. Clock count is performed by decrementing bits 21 and 23 every 1/60th of a second or by decrementing bits 21, 22 by every 1/50th of a second, depending on line frequency (in either case, the effect is equivalent to reducing the timer by 1 in bit 23 every 1/300th of a second). The word is treated as a signed binary operand and follows the rules of fixed-point arithmetic. An interrupt condition is effected when the clock goes from positive to negative.

The clock does job accounting by measuring the duration of time for each job, monitors interrupt to prevent a runaway job from controlling the system, records time of day, and monitors a communication network on regular intervals—say, every minute or hour.

Illustration

Word 80 is set equal to:

0 000 0000 0000 0000 0000 1001 1000 0010

How much time will elapse before the clock will effect interrupt?

This is equivalent to $(982)_{16} = (2434)_{10}$

The clock decrements by $(256)_{10}$ every 3.3ms.

Thus, in 33ms interrupt is effected.

Exercise 5

What constant could one store in word 80 so that the clock will effect interrupt every 5 minutes?

APPENDIX A

INSTRUCTIONS IN ORDER BY MNEMONIC

Instruction	Mnemonic Code	Operation Code	Operand Field Format
Add	A	5A	R1, D2(X2, B2)
Add Normalized, Long	AD	6A	R1, D2(X2, B2)
Add Normalized, Long	ADR	2A	R1, R2
Add Normalized, Short	AE	7A	R1, D2(X2, B2)
Add Normalized, Short	AER	3A	R1, R2
Add Half-Word	AH	4A	R1, D2(X2, B2)
Add Logical	AL	5E	R1, D2(X2, B2)
Add Logical	ALR	1E	R1, R2
Add Decimal	AP	FA	D1(L1, B1), D2(L2, B2)
Add	AR	1A	R1, R2
Add Unnormalized, Short	AU	7E	R1, D2(X2, B2)
Add Unnormalized, Short	AUR	3E	R1, R2
Add Unnormalized, Long	AW	6E	R1, D2(X2, B2)
Add Unnormalized, Long	AWR	2E	R1, R2
Branch and Link	BAL	45	R1, D2(X2, B2)
Branch and Link	BALR	05	R1, R2
Branch on Condition	BC	47	M, D2(X2, B2)
Branch on Condition	BCR	07	M, R2
Branch on Count	BCT	46	R1, D2(X2, B2)
Branch on Count	BCTR	06	R1, R2
Branch on Index High	BXH	86	R1, R3, D2(B2)
Branch on Index Low or Equal	BXLE	87	R1, R3, D2(B2)
Compare Algebraic	C	59	R1, D2(X2, B2)
Compare, Long	CD	69	R1, D2(X2, B2)
Compare, Long	CDR	29	R1, R2
Compare, Short	CE	79	R1, D2(X2, B2)
Compare, Short	CER	39	R1, R2
Compare Half-Word	CH	49	R1, D2(X2, B2)
Check Channel	CKC	9F	D1(B1)
Compare Logical	CL	55	R1, D2(X2, B2)
Compare Logical	CLC	D5	D1(L, B1), D2(B2)
Compare Logical Immediate	CLI	95	D1(B1), I
Compare Logical	CLR	15	R1, R2
Compare Decimal	CP	F9	D1(L1, B1), D2(L2, B2)
Compare Algebraic	CR	19	R1, R2
Convert to Binary	CVB	4F	R1, D2(X2, B2)
Convert to Decimal	CVD	4E	R1, D2(X2, B2)
Divide	D	5D	R1, D2(X2, B2)
Divide, Long	DD	6D	R1, D2(X2, B2)
Divide, Long	DDR	2D	R1, R2
Divide, Short	DE	7D	R1, D2(X2, B2)
Divide,. Short	DER	3D	R1, R2
Divide Decimal	DP	FD	D1(L1, B1), D2(L2, B2)
Divide	DR	1D	R1, R2
Edit	ED	DE	D1(L, B1), D2(B2)
Edit and Mark	EDMK	DF	D1(L, B1), D2(B2)
Execute	EX	44	R1, D2(X2, B2)
Halve, Long	HDR	24	R1, R2
Halt Device	HDV	9E	D1(B1)
Halve, Short	HER	34	R1, R2
Idle	IDLE	80	
Insert Character	IC	43	R1, D2(X2, B2)
Insert Storage Key	ISK	09	R1, R2
Load	L	58	R1, D2(X2, B2)
Load Address	LA	41	R1, D2(X2, B2)
Load Complement, Long	LCDR	23	R1, R2
Load Complement, Short	LCER	33	R1, R2
Load Complement	LCR	13	R1, R2
Load, Long	LD	68	R1, D2(X2, B2)

Instruction	Mnemonic Code	Operation Code	Operand Field Format
Load, Long	LDR	28	R1, R2
Load, Short	LE	78	R1, D2(X2, B2)
Load, Short	LER	38	R1, R2
Load Halfword	LH	48	R1, D2(X2, B2)
Load Multiple	LM	98	R1, R3, D2(B2)
Load Negative, Long	LNDR	21	R1, R2
Load Negative, Short	LNER	31	R1, R2
Load Negative	LNR	11	R1, R2
Load Positive, Long	LPDR	20	R1, R2
Load Positive, Short	LPER	30	R1, R2
Load Positive	LPR	10	R1, R2
Load	LR	18	R1, R2
Load Scratch Pad	LSP	D8	D1(L, B1)D2(B2)
Load and Test, Long	LTDR	22	R1, R2
Load and Test, Short	LTER	32	R1, R2
Load and Test	LTR	12	R1, R2
Multiply	M	5C	R1, D2(X2, B2)
Multiply, Long	MD	6C	R1, D2(X2, B2)
Multiply, Long	MDR	2C	R1, R2
Multiply, Short	ME	7C	R1, D2(X2, B2)
Multiply, Short	MER	3C	R1, R2
Multiply Halfword	MH	4C	R1, D2(X2, B2)
Multiply Decimal	MP	FC	D1(L1, B1), D2(L2, B2)
Multiply	MR	1C	R1, D2(X2, B2)
Move Characters	MVC	D2	D1(L, B1), D2(B2)
Move Immediate	MVI	92	D1(B1), S2
Move Numeric	MVN	D1	D1(L, B1), D2(B2)
Move with Offset	MVO	F1	D1(L1, B1), D2(L2, B2)
Move Zones	MVZ	D3	D1(L, B1), D2(B2)
AND Logical	N	54	R1, D2(X2, B2)
AND Logical	NC	D4	D1(L, B1), D2(B2)
AND Logical Immediate	NI	94	D1(B1), I
AND Logical	NR	14	R1, R2
OR Logical	O	56	R1, D2(X2, B2)
OR Logical	OC	D6	D1(L, B1), D2(B2)
OR Logical Immediate	OI	96	D1(B1), I
OR Logical	OR	16	R1, R2
Pack	PACK	F2	D1(L1, B1), D2(L2, B2)
Program Control	PCTL	82	D1(B1)
Read Direct	RDD	85	D1(B1), I
Subtract	S	5B	R1, D2(X2, B2)
Subtract Normalized, Long	SD	6B	R1, D2(X2, B2)
Subtract Normalized, Long	SDR	2B	R1, R2
Start Devices	SDV	9C	D1(B1)
Subtract Normalized, Short	SE	7B	R1, D2(X2, B2)
Subtract Normalized, Short	SER	3B	R1, R2
Subtract Halfword	SH	4B	R1, D2(X2, B2)
Subtract Logical	SL	5F	R1, D2(X2, B2)
Shift Left Single Algebraic	SLA	8B	R1, D2(B2)
Shift Left Double Algebraic	SLDA	8F	R1, D2(B2)
Shift Left Double Logical	SLDL	8D	R1, D2(B2)
Shift Left Single Logical	SLL	89	R1, D2(B2)
Subtract Logical	SLR	1F	R1, R2
Subtract Decimal	SP	FB	D1(L1, B1), D2(L2, B2)
Set Program Mask	SPM	04	R1
Subtract	SR	1B	R1, R2
Shift Right Single Algebraic	SRA	8A	R1, D2(B2)
Shift Right Double Algebraic	SRDA	8E	R1, D2(B2)
Shift Right Double Logical	SRDL	8C	R1, D2(B2)
Shift Right Single Logical	SRL	88	R1, D2(B2)
Set Storage Key	SSK	08	R1, R2

APPENDIX A (Continued)

Instruction	Mnemonic Code	Operation Code	Operand Field Format
Store Scratch Pad	SSP	Do	D1(L1, B1), D2(B2)
Store	ST	50	R1, D2(X2, B2)
Store Character	STC	42	R1, D2(X2, B2)
Store, Long	STD	60	R1, D2(X2, B2)
Store, Short	STE	70	R1, D2(X2, B2)
Store Halfword	STH	40	R1, D2(X2, B2)
Store Multiple	STM	90	R1, R3, D2(B2)
Subtract Unnormalized Short	SU	7F	R1, D2(X2, B2)
Subtract Unnormalized	SUR	3F	R1, R2
Supervisor Call	SVC	0A	I
Subtract Unnormalized Long	SW	6F	R1, D2(X2, B2)
Subtract Unnormalized Long	SWR	2F	R1, R2
Test Device	TDV	9D	D1(B1)
Test Under Mask	TM	91	D1(B1), I
Translate	TR	DC	D1(L, B1), D2(B2)
Translate and Test	TRT	DD	D1(L, B1), D2(B2)
Unpack	UNPK	F3	D1(L1, B1), D2(L2, B2)
Write Direct	WRD	84	D1(B1), I
Exclusive OR	X	57	R1, D2(X2, B2)
Exclusive OR	SC	D7	D1(L, B1), D2(B2)
Exclusive OR, Immediate	XI	97	D1(B1), I
Exclusive OR	XR	17	R1, R2
Zero and Add Decimal	ZAP	F8	D1(L1, B1), D2(L2, B2)

APPENDIX B
CONDITION CODE

Operation	0	1	2	3
Result of:				
Arithmetic[1,4]	0	-	+	overflow
And, Or, Xor	0	≠ 0		
ED, EDMK	0	-	+	
Shift Left[3]	0	-	+	overflow
Load complement	0	-	+	overflow
Load positive	0		+	overflow
Load negative	0	-		overflow
Load and test	0	-	+	
Logical arithmetic[2]	0, NC[5]	≠0, NC	0, C	≠0, C
Shift right[3]	0	-	+	
Comparisons:				
Operand 1 is	=	<	>	
TRT	all table bytes 0	table byte ≠0 found	last table byte ≠0	
Set Program Mask	Reset	Reset	Reset	Reset
TM: Selected Bits	Zeros (or Mask is zero)	Mixed		Ones

Notes

1. Overflow in floating point arithmetic is exponent overflow.
2. In logical arithmetic, C indicates a carry out of sign position; NC indicates no carry.
3. Only algebraic shifts set the condition code.
4. Multiply, Divide, and Halve do not set the condition code.
5. Add only.