IBM TECHNICAL INFORMATION EXCHANGE

TIE 5-0021
December 22, 1964
190 pages

XT

IBM 1401, 1440 AND 1460 PROGRAMMING AND OPERATING TECHNIQUES

Mr. Jack Melnick
150 Grand Street- Basement
White Plains, N. Y.

Distributed by
DPD Program Information Department
IBM Corporation
112 East Post Road
White Plains, New York

IBM 1401, 1440 and 1460 Programming and Operating Techniques

Page Key

# of Pages	
3	Unnumbered pages in front (title, disclaimer, abstract)
1	Contents
6	Section A (A-1 thru A-6)
71	Section B (B-0 thru B-69)
13	Section C (C-1 thru C-13)
15	Section D (D-0 thru D-14)
10	Section E (E-0 thru E-9)
11	Section F (F-0 thru F-10)
7	Section G (G-0 thru G-6)
8	Section H (H-0 thru H-7)
23	Section I (I-0 thru I-22)
20	Section J (J-0 thru J-19)
188	

IBM 1401, 1440 and 1460

PROGRAMMING AND OPERATING TECHNIQUES

A. Elaine Taylor
Advisory Systems Specialist
150 Grand Street
White Plains, New York

Maurice D. Howe
Marketing Publications
Washington Ave. Lab
Dept. 293/Bldg. 630-1
Endicott, New York

Jack Melnick
Manager, Field Techniques Development Projects
150 Grand Street
White Plains, New York

August 18, 1964

To the best of our knowledge, the contents of our work entitled "IBM 1401, 1440 and 1460 PROGRAMMING AND OPERATING TECHNIQUES" is free of any proprietary, secret or confidential information belonging to a person or organization outside the IBM company. Since this paper is a collection of techniques, we have used the work of others and have obtained permission to do so, where necessary.

A. Elaine Taylor Maurice D. Howe Jack Melnick

August 18, 1964

IBM 1401, 1440 and 1460

PROGRAMMING AND OPERATING TECHNIQUES

This paper contains programming and operating tips for the IBM 1401, 1440 and 1460 Data Processing Systems, and is intended to supplement the System Operation Reference Manuals, Special Features Manuals, and other manuals of the 1401/1440/1460 SRL series. These tips and pointers can be considered the 1400 Series equivalent of the old unit record "Principles of Operation Bulletins".

8/18/64

Direct Inquiries to:

Jack Melnick
150 Grand Street
White Plains, New York
WH9-1900 X4517

OVERALL TABLE OF CONTENTS

Available Now

Future Sections

(A) General System Techniques Information

(A-1) Standard BCD Interchange Code

The Standard BCD Interchange Code defines for the IBM Corporation a standard 64-character set for the IBM 1401, 1440, 1410, 1460, 7040, and 7044 Data Processing Systems. The code provides compatibility of data for interchange among all systems using this standard. The standard provides a consistent definition of:

1. IBM Card Code
2. IBM BCD Magnetic Tape Code
3. Relation between these codes and printed symbols (graphics)
4. Relation between these codes and machine control codes
5. Collating sequence of code elements
6. Two subsets of alternate graphics

In addition, the standard provides uniform graphics for publications. Existing published material will be changed to reflect the standard BCD interchange code.

Figure A-1 is a chart of the standard BCD interchange code. Column 2 shows the graphics for the 64 code elements. The equivalent card and BCD codes are shown by columns 3 and 4. The collating sequence of the 64 code elements is indicated in column 1 by a collating number which runs from 00 (low) to 63 (high).

1 Collating Number	2 Graphics	3 Card Code	4 BCD Code B	A	8	4	2	1	5 Spec Sign.	6 Char. at d Branch	I/O	7 IBM 1401 Op Code
00	Blank	No Punches	No Bits									
01	.	12 3 8	B	A	8		2	1				Halt.
02	ɣ(	12 4 8	B	A	8	4					wr. w/m	Clear word mark.
03	⌈	12 5 8	B	A	8	4		1				
04	≤	12 6 8	B	A	8	4	2					
05	≠	12 7 8	B	A	8	4	2	1	GM			
06	& +	12	B	A								
07	$	11 3 8	B		8		2	1				
08	*	11 4 8	B		8	4				Inq. Clear		
09	)	11 5 8	B		8	4		1				
10	;	11 6 8	B		8	4	2					
11	Δ	11 7 8	B		8	4	2	1	MC			
12	-	11	B									
13	/	0 1		A				1		Uneq-Comp.		Clear storage.
14	¢	0 3 8		A	8		2	1				Set word mark.
15	% (	0 4 8		A	8	4				Proc. Check		Divide
16	¥	0 5 8		A	8	4		1	WS			
17	\	0 6 8		A	8	4	2					
18	m	0 7 8		A	8	4	2	1	SM			
19	ъ	2 8		A					SB			
20	# =	3 8			8		2	1				Modify Address
21	@ !	4 8			8	4				carr. char. 12		Multiply
22	:	5 8			8	4		1				
23	>	6 8			8	4	2					
24	∫	7 8			8	4	2	1	TM			
25	?	12 0	B	A	8		2		PZ	RD I/O err.		Zero and add.
26	A	12 1	B	A				1		last card sw./bin-decode		Add.
27	B	12 2	B	A			2			sense sw. B/BSP tape		Branch.
28	C	12 3	B	A			2	1		sense sw. C/col. binary		Compare.
29	D	12 4	B	A		4				sense sw. D		Move digit only.
30	E	12 5	B	A		4		1		sense sw. E/skip+erase		Edit.
31	F	12 6	B	A		4	2			sense sw. F		Carriage control.
32	G	12 7	B	A		4	2	1		sense sw. G		

Figure A-1
Standard BCD Interchange Code

1 Collating Number	2 Graphics	3 Card Code	4 BCD Code B	A	8	4	2	1	5 Spec. Sign.	6 Char. at d Branch	I/O	7 IBM 1401 Op Code
33	H	12 8	B	A	8					rder busy		Store B-register
34	I	12 9	B	A	8			1		pch busy		
35	¦	11 0	B		8		2		MZ	pch I/O err,		Zero and subtract
36	J	11 1	B					1		tape or I/O busy		
37	K	11 2	B				2			end of reel		Stacker select
38	L	11 3	B				2	1		tape error		Load
39	M	11 4	B			4					write T/M	Move
40	N	11 5	B			4		1		disk access inop/		No operation
41	O	11 6	B			4	2					
42	P	11 7	B			4	2	1		printer busy		Move record
43	Q	11 8	B		8					inq. req.	rd. pch. feed rwd. tape	Store A-register
44	R	11 9	B		8			1		carriage busy	R	
45	≠	0 2 8		A	8		2		RM	printer I/O		
46	S	0 2		A			2			eq. comp.		Subtract
47	T	0 3		A			2	1		low comp.		
48	U	0 4		A		4				high comp. /rwd+unl tape		Control unit
49	V	0 5		A		4		1		disk rd. bl. ck.		Branch if zone or w/m
50	W	0 6		A		4	2			disk wrong lgth.		Branch if bit equal
51	X	0 7		A		4	2	1		disk uneq. addr.		Expand compressed tape
52	Y	0 8		A	8					any disk error		Move zone only
53	Z	0 9		A	8			1		arith of.		Move & suppress zeros
54	0	0			8		2			1404 err/rd. 1404		
55	1	1						1				Read a card
56	2	2					2					Print
57	3	3					2	1				Print and read
58	4	4				4						Punch
59	5	5				4		1				Read and punch
60	6	6				4	2					Print and punch
61	7	7				4	2	1				Print, read and punch
62	8	8			8							Start read feed
63	9	9			8			1		carr. char. #9		Start punch feed

Figure A-1
Standard BCD Interchange Code

(A-2) Word Mark Control in Data Movement

The function of the word mark is illustrated in Figure A-5. Operation codes which either do not require word marks to end them, or do not affect them, are not listed (such as clear storage, punch, set word mark instructions, etc.).

Word Mark Control in Data Movement

Instruction			Word Mark Stops		Word Marks are erased	A-field Word mark Transferred
Op Code	Mnemonic	Function	Data transfer only	Data transfer and operation		
A	A	Add	A	B	None	No
S	S	Substract	A	B	None	No
ȯ	ZA	Zero and add	A	B	None	No
ō	ZS	Zero and subtract	A	B	None	No
@	M	Multiply	B	B(multiplier)	None	No
%	D	Divide	A	None(sign of dividend)	None	No
C	C	Compare	B	B	None	No
M	MCW MLC	Move	Either	Either	None	No
L	LCA MLCWA	Load	A	A	Any in B-field	Yes
D	MN MLNS	Move Num	None	None	None	No
Y	MZ MLZS	Move Zone	None	None	None	No
Z	MCS	Move and zero supp.	A	A	Any in B-field	No
E	MCE	Move and edit	A	B	Any in B-field	No
P	MCM MRCM	Move record	A ∓ or A ∓ (from left to right)	A ∓ or A ∓	None	No
X	MIZ	Expand	A (from left to right)	A ∓	None	No
M	RT	Read tape	IRG or ∓ in B	IRG	NONE	IRG creates ∓
L	RTW	Read tape	IRG or ∓ in B	IRG	B	Yes(from tape)
M	WT	Write tape	∓ in B-field	∓ in B-field	None	No
L	WTW	Write tape	∓ in B-field	∓ in B-field	None	Yes(as Word Separator char) except ∓

Figure A-5

(B) Subroutines and Subroutine Techniques

(B) Subroutines and Subroutine Techniques (cont'd.)

(B-1) Address Coding and Decoding

IBM **1401/1440/1460 ADDRESS TRANSLATION CARD**

Locate on the hundreds axis the first character of the coded address and the line in which it appears and on the units axis locate the last character of the coded address and the column in which it appears. The subscript for each character located is the numerical value for the corresponding address position and the point of intersection is the thousands designation. Thus the machine-coded address BPY is 72x8. Indexing is indicated if the middle character of the coded address is an alphabetic or special one. Locate the character in the character set labeled hundreds; to the far left of that line is the index location. BPY is location 7278 indexed by location 2. By reversing the procedures, numerical addresses can be translated into machine-coded addresses.
Additional examples: PYH= 14788 1; 5642 3=WDS; P19=2719 none; A6Z=7169 none; 15123 none=A2C; CC4=3334 3.

UNITS →

HUNDREDS ↓

IND. LOC.	?$_0$ A$_1$ B$_2$ C$_3$ D$_4$ E$_5$ F$_6$ G$_7$ H$_8$ I$_9$	!$_0$ J$_1$ K$_2$ L$_3$ M$_4$ N$_5$ O$_6$ P$_7$ Q$_8$ R$_9$	‡$_0$ /$_1$ S$_2$ T$_3$ U$_4$ V$_5$ W$_6$ X$_7$ Y$_8$ Z$_9$	0$_0$ 1$_1$ 2$_2$ 3$_3$ 4$_4$ 5$_5$ 6$_6$ 7$_7$ 8$_8$ 9$_9$
3	?$_0$ A$_1$ B$_2$ C$_3$ D$_4$ E$_5$ F$_6$ G$_7$ H$_8$ I$_9$	15xxx	11xxx	7xxx	3xxx
2	!$_0$ J$_1$ K$_2$ L$_3$ M$_4$ N$_5$ O$_6$ P$_7$ Q$_8$ R$_9$	14xxx	10xxx	6xxx	2xxx
1	‡$_0$ /$_1$ S$_2$ T$_3$ U$_4$ V$_5$ W$_6$ X$_7$ Y$_8$ Z$_9$	13xxx	9xxx	5xxx	1xxx
NONE	0$_0$ 1$_1$ 2$_2$ 3$_3$ 4$_4$ 5$_5$ 6$_6$ 7$_7$ 8$_8$ 9$_9$	12xxx	8xxx	4xxx	0xxx

(B-2) Address Modification without the Modify Address Op Code

To increase any 1401/1440/1460 address, the Add Op may be used. Set a word mark in the hundreds position of the address to be modified. Convert the modification factor to a 3-digit number: 1204 is actual machine address S04. Add this 3-digit factor to the address to be modified. Clear the Word Mark set in step 1. The resultant answer will be a valid address. Remember, however, that an undetected wrap-around may have occurred.

To decrease an address, convert the modification factor to the 16,000 complement, and use the Add Op. Thus: to decrease the address G67 (3767) by -003, convert address factor -003 to 15,997. This will be actual machine address I9G. Add I9G to G67, and the result is G64 (3764). Arithmetic overflow controls the zone bits in the hundreds position of the modified address.

Note that any indexing bits over the tens position of the B-field will be stripped.

If indexing bits are present in the modification factor, they will be ignored. If indexing bits are present in the original address to be modified, they will be removed. If the indexing bits will be required, they can be removed prior to the Add Op by a Move Zone to a save area and replaced after the modification.

This method of address modification is not necessary for addresses over 4,000 since Modify Address is standard on machines with more than 4,000 positions of core storage.

(B-3) Address Register Contents

The original contents of the B-STAR (B-Storage Address Register) are not disturbed when the A-STAR is read into, when using the following op codes as single address instructions (op code and an A-address):

M Move characters to a word mark in either field.

L Load characters to the A-field word mark.

H Store B-STAR contents.

Q Store A-STAR contents.

The contents of the B-STAR are destroyed when the A-STAR is used for the next op, except when chaining, using op codes only.

In machines having the Advanced Programming Feature, the contents of the I-STAR will be transferred to the B-STAR, after any successful branch op. The first instruction of the branched-to routine can then be a Store B-STAR op (SBR), so that the address of the NSI (Next Sequential Instruction) following the branch can be retained to provide effective routine linkage.

(B-4) Clear Storage Following Print, Read, or Punch Operations

The print area in the 1401/60 is normally cleared as follows:

Actual	Mnemonic
2	W
/332	CS 332
/	CS

The print area can be cleared by using the print op code (2), followed by two chained CS (Clear Storage) op codes, thus:

Actual	Mnemonic
2	W
/	CS
/	CS

At the end of the print op, the B-STAR will contain address 335 or 333 if print storage is installed. By taking advantage of this fact, 3 core positions can be saved every time the print area is to be cleared.

When this method is used, the program must originate at core location 336 or 334 with print storage, instead of the usual 333. The program may still start at location 333, if the instruction or constant at location 333 is to be used before the first print op, and never referred to again.

This method of clearing core may be used following other input/output op codes. Figure B-3 shows ending addresses after 1402/1403 operations.

I/O Op	Inst.	A *	B *	B * with print storage
Read	1	xxx	081	
Rd/Br	1 707	707	081	
Print	2	xxx	335	333
Pr/Br	2 623	623	335	333
Pr/Wm	2 ⌑	yyy	335	333
Pr/Wm/Br	2 795 ⌑	795	335	333
Pr/Rd	3	xxx	081	
Pr/Rd/Br	3 650	650	081	
Punch	4	xxx	181	
Pch/Br	4 925	925	181	
Rd/Pch	5	xxx	181	
Rd/Pch/Br	5 893	893	181	
Pr/Pch	6	xxx	181	
Pr/Pch/Br	6 724	724	181	
Pr/Rd/Pch	7	xxx	181	
Pr/Rd/Pch/Br	7 392	392	181	
Pfr	4 R	yyy	181	
Pfr/Br	4 823R	823	181	
Pfr/Pr/Br	6 R	yyy	181	
Pfr/Pr/Br	6 823R	823	181	

xxx denotes previous setting of A-address register.
yyy denotes the d-character and blanks in the units and tens position.

Figure B-3
Ending Address After 1402/1403 Operations

(B- 5) Conversion of 5-Digit Addresses to 3-Digit Addresses

Refer to program (Figure B-4). .

The first overflow gives an A-zone in the high order position of HOLDAR (HOLDAR-4). This zone is determined by the two digits that must fit in this one position as: 10 is ∓ , and 19 is Z. As long as the digit does not change by adding 96, no overflow is indicated. When the zone generated changes to a B-zone, overflow is indicated again. Thus, addresses in the range 0-3999 cause no overflow and therefore no zone in the units position of the address. The range 4000-7999, one overflow occurs giving an A-zone in the units position of the converted address. In the range 8000-11999; two overflows; B-zone in the units position. In the range 12000-15999; three overflows; AB-zone in the units position.

This program (Figure B-4) requires very little storage and is easy to program and to understand. It can be used in any application where address conversion is necessary.

Step Number	Label	Op	Operand	Comments
1	CNVTO3	BAV	*+001	Reset overflow
2		A	@96@, HOLDAR-003	Generate 1-3 overflows
3		BAV	CNVTO3+005	
4		MZ	HOLDAR-004, HOLDAR #5	Overflow to zone units pos.
5		MN	HOLDAR-003, *+004	Move digit (6, 7, 8, 9)
6		MZ	ZONE, HOLDAR-002	Move corresponding zone to hundreds position.
		ORG	909	
	ZONE	DCW	@25KB@	

These statements might assemble as:

Step Number	INST ADD	INSTRUCTION	COMMENTS
1	601	B 606 Z	Reset Overflow Indicator
2	606	A 702 802	Generate 1 to 3 overflows.
3	613	B 606 Z	Branch to add Op if overflow.
4	618	Y 801 805	Use overflow for units sign.
5	625	D 802 635	Move digit (6, 7, 8, or 9).
6	632	Y 909 803	Move corresponding zone to hundreds positon.
	639	NSI	
	DCW's		
	Units Add.	Factor	Comments
	702	96	Alpha Literal.
	805	00000	5 position hold area into which the address to be converted is program-loaded.
	909	2SKB	Literal located so that the zones can be used. Units position (B) must be in location ending in nine

Figure B-4 Address Conversion Routine.

HOLDAR	Step 2	Step 3	Step 4	Step 5 gives Address	Step 6 zone	Result	NSI
00100	+96 = 96100	No ovfl	96100	906	Blank	96100 or 100	
01000	+96 = 97000	No ovfl	97100	907	A-Zone	97‡00 or ‡00	
03000	+96 = 99000	No ovfl.	99100	909	AB-Zone	99000 or 000	
04000	+96 = ‡0000	Ovfl					
	+96 = Z6000	No ovfl	Z600‡	906	Blank zone	Z600‡ or 00‡	
07000	+96 = ‡3000	Ovfl					
	+96 = Z9000	No ovfl	Z900‡	909	AB-Zone	Z900‡ or 00‡	
08000	+96 = ‡4000	Ovfl					
	+96 = 00000	Ovfl					
	+96 = R6000	No ovfl	R600$\bar{0}$	906	Blank zone	R600$\bar{0}$ or 00$\bar{0}$	
11000	+96 = ‡7000	Ovfl					
	+96 = 03000	Ovfl					
	+96 = R9000	No ovfl	R900$\bar{0}$	909	AB-Zone	R900$\bar{0}$ or 00$\bar{0}$	
12000	+96 = ‡8000	Ovfl					
	+96 = J4000	Ovfl					
	+96 = 00000	Ovfl					
	+96 = I6000	No ovfl	I600$\bar{0}$	906	Blank zone	I600$\bar{0}$ or 00$\bar{0}$	
15000	+96 = /1000	Ovfl					
	+96 = Q7000	Ovfl					
	+96 = 03000	Ovfl					
	+96 = I9000	No ovfl	I900$\bar{0}$	909	AB-Zone	I900$\bar{0}$ or 00$\bar{0}$	

Figure B-5 Function of Address Conversion Subroutine

(B-6) Programmed "Wrap-Around"

If storage location 000 is addressed with any instruction that decrements an A- or B-address register, the processing system will wrap-around to its high-core storage address (3999, 7999, 11999 or 15,999).

To determine the core storage size of a particular machine as the program is being run, use the Clear Storage (/) op, followed immediately by the Store B-Register (H) op, thus:

```
CS  O
SBR AAA
```

The object machine storage size will be stored in the core location represented by the notation (AAA). The clear storage instruction does not cause a wrap-around error.

Do not attempt to address storage location 15,999 (or maximum storage location of your particular system), with any op code which increments the A- or B-address, since the system will wrap-around to its low-core storage address (000).

The op codes which will cause this error are the Move Record Op, or any of the serial ops, such as magnetic tape, serial I/O, serial readers, etc.

(B-7) Using Incoming Data to Modify Instructions

Techniques to use incoming data to directly modify instructions, as opposed to the more standard approach of testing, branching, and use of subroutines, follow:

1. Consider, for example, a program in which a transaction or accounting code on data documents is to affect one of a series of accumulators. The thirty-six codes are the single characters, letters A through Z, and the digits 0-9, in this sequence.

   Assign ten position accumulators, of which there will be thirty-six, with units positions in core locations 3649, 3659, etc. up to 3999. Now rather than testing for each code individually, use one instruction (the object instruction) and modify its B-operand, which specifies the location to be changed. Note that the numeric values of each transaction code, (the letters A-I), correspond to the relative sequence of the accumulator which they are to affect. The same is true for letters J-R if the value of 9 is added, for the letters S-Z if the value of 17 is added, and for the numbers zero to nine if 27 is added. These would be the base numbers to which would be added the digital value of the code character.

Code Character	Base + Digit	Accumulator Sequence
A = (1)	0 + 1	= 1
I = (9)	0 + 9	= 9
J = (1)	9 + 1	= 10
R = (9)	9 + 9	= 18
S = (2)	17 + 2	= 19
Z = (9)	17 + 9	= 26
0 = (0)	27 + 0	= 27
9 = (9)	27 + 9	= 36

Programming can take direct advantage of this logical relationship. The routine (Figure B-6) would consist of instructions to analyze the zone structure of the transaction code, branching to add the corresponding zone value to the tens position of a three-position constant, which has the initial value of machine address 3639. The numeric portion of the code is then added to the tens position of the constant so that the resulting value which will be in increments of ten is one of thirty-six machine addresses, the units position of the correct accumulator. The constant then replaces the B-operand. Total core requirements are 105 locations, in contrast to 684, using standard programming.

LABEL	OP	A-ADDR	B-ADDR	d	
	BWZ	ADD 9	CODE	K	CK Eleven Zone
	BWZ	ADD 17	CODE	X	CK Zero Zone
	BWZ	ADD 27	CODE	2	CK No Zone
RETURN	MZ	BLANK	CODE		Dezone Numeric
	A	CODE	WKAREA-001		Add Numeric
	MCW	WKAREA	OBJECT+006		
OBJECT	A	AMOUNT	0000		
	MCW	RESET	WKAREA		
TO MAIN ROUTINE					
ADD 9	A	NUM 9	WKAREA-001		
	B	RETURN			
ADD 17	A	NUM 17	WKAREA-001		
	B	RETURN			
ADD 27	A	NUM 27	WKAREA-001		
	B	RETURN+007			

Figure B-6.1
Transaction Code Test

```
SIZE    LABEL      OP
03      WKAREA     DCW     *       F39
03      RESET      DCW     *       F39
02      NUM 9      DCW     *       09
02      NUM 17     DCW     *       17
02      NUM 27     DCW     *       27
01      BLANK      DCW     *
        CODE       DS      0070
        AMOUNT     DS      0060
```

Figure B-6.1 (cont'd)
Transaction Code Test

2. For areas of different size, for example, seven positions, the sum of the zone and underpunch might be expanded seven times by successive addition and subtraction, with modification then taking place at the units position of the operand (or index register, if available). Total core requirements are approximately 146 with this technique as opposed to 684.

3. Consider another situation of just six accumulating areas. Each incoming code may add to, subtract from, or bypass an area; any code affects on the average, half of the accumulators. The solution here could result in one object instruction for each area, with data analyzing instructions choosing the operation codes in advance.

   Each code would have an associated six-position constant, containing the required machine language op codes in sequence. After the location of these constants has been assigned, the data code is used to develop the machine address of the correct constant. Other instructions insert the op codes from the constant to the six object instructions. The A-operand of the first instruction contains the address of the selected DCW (Figure B-7.2).

LABEL	OP	A-ADDR	B-ADDR	COMMENTS
OBJECT	MCW	0000	AREAX	
	MCW	AREAX-005	INSTRA	INSTRA through INSTRF are the six object instructions.
	MCW	AREAX-004	INSTRB	
	MCW	AREAX-003	INSTRC	
	MCW	AREAX-002	INSTRD	
	MCW	AREAX-001	INSTRE	
	MCW	AREAX	INSTRF	

06	DCW	*ASNNAN		Typical DCW with six machine lang. op codes
06	DCW	*NNSSAN		
	---			
06 AREAX	DCW	*		

Figure B-7.2
Op Code Insertion

Storage locations in this example are approximately 125 for instructions, 230 for constants for a total of 355. Standard programming, involving one test, three arithmetic and one branch instruction for each of thirty-six codes, would be about 1200 positions. Of course, the constant could contain any element, or combination of elements, that could comprise an instruction.

(B-8) Double Identification Halts

Often it is useful to program halt ops to serve multiple functions. For instance, a program might contain a halt #1111 to tell the operator to insert a tax year in a particular storage location, such as: H 1111, TXYR, where the label TXYR will refer to the units position of the tax year field in the assembled program. This instruction might appear in the post list as: . /11 908.

(B-9) Programmed Halt Numbering

Standardization of programmed halt numbering facilitates job operations.

By using specific halt numbers for specific halt conditions, program coding and job operation is simplified. The halt instruction can be used to indicate common halt conditions, such as: out-of-sequence, tape error, reader erros, end-of-job, etc.

Halt Types

# Positions	Instructions
1	H
2	H 2
3	NOP 2 H
4	H 222
5	NOP 222 H
7	H 111 222

The 2 and 3 position halt instruction provides one position for halt indications, while the 5 and 7 position halt instruction provide three positions. In the case of the 1, 2, 3, 5 and 7 position halts, the NSI will be executed when the start key is pressed. The 4 position halt will cause a branch to the location in the A-STAR.

(B-9.1) Dead-End Halts

Some error conditions demand that the application in progress be terminated. It will have been determined beforehand that further attempts to process will be useless. Typical examples of this follows:

```
FATAL   H     FATAL      (The address of the label
                         FATAL can be standard-
                         ized for all programs.)
```

or

```
FATAL   H     888,888
        B     FATAL
```


(B-10) Programmed Halt, Two-Position

The 2-position halt instruction has the same characteristics as the seven position halt; the operand is displayed in both the A- and B-address registers. When the start key is pressed, the next sequential instruction will be executed. Five core storage positions are saved. It is coded as follows:

```
HALT9   H
        DC    @9@
```

or:

```
HALT9   DCW   @.9@
```

Note:

When using this technique, the digit modifier will be dropped when assembling is done with autocoder, if written in symbolic SPS format. Therefore, it should be coded in actual (.9).

(B-11) Iteration Controllers

Three methods of counting iterations (repetitive operations) are:

1. Branch if Character Equal
2. Compare
3. Rotary Switch

The Branch if Character Equal method (Figure B-8) is limited to a maximum of 10 iterations, but has the advantage of using a minimum amount of core. It is easily modified if iteration control is to vary during the job

A variation of the Branch if Character Equal method can be used to count iterations over 10, by first branching on a specific character in the 10's (hundreds or thousands, etc.) position of the count field. When this test is positive, modify the Branch if Character Equal instruction to test the next lower digit of the count field. Usually, both the d-modifier and the B- (testing) address will have to be changed. The final test positive will provide a branch to the RESET sub-subroutine, and a branch out of the subroutine to the NSI.

The Compare Op method (Figure B-9) can be used in any general iteration count. It is easy to program, but is more expensive of core storage than the previous method.

The Rotary Switch method (Figure B-10)uses less core than the compare equal method for iteration counts of 2 to 6.

Any of these count-subroutines can be used wherever a specific or program-alterable count device is required, such as tape blocking, table lookup, multiply subroutines, etc.

LABEL	Op	A-Add	B-Add	d	Count	COMMENTS
						Follow steps of subroutine.
ADDITR	A	ADDITR	COUNT		7	UsesADDITR Op for constant.
	B	RESET	COUNT	d	8	d-modifier any plus-zoned character.
	B	SUBROU			4	Branch back to subroutine.
RESET	ZA	RESET	COUNT		7	Uses ZA Op as DCW
	B	NSI			4	Branch out of subroutine.
COUNT	DCW	* +0			1	Must be plus zero.
			Total Positions		31	

Figure B-8 Branch if character Equal Iteration Control Subroutine.

LABEL	Op	A-Add	B-Add	d	Count	COMMENTS
ADDITR	A	ADDITR	COUNT		7	Uses Op-code for constant.
	C	CONSTN	COUNT		7	Compare constant factor to count.
	B	RESET		S	5	Branch if compare equal.
	B	SUBROU			4	Branch back to subroutine.
RESET	ZA	RESET	COUNT		7	Uses ZA Op-code for constant.
	B	NSI			4	Branch out of subroutine.
COUNT	DCW	*	000 etc.		n	Must be zoned plus, n positions long.
CONSTN	DCW	*	000 etc.		n	Same length as COUNT.
			Total:		34 plus 2n	

Figure B-9 Compare Equal Iteration Control Subroutine.

LABEL	Op	A-Add	B-Add	COUNT	COMMENTS
					Follows steps of subroutine.
STEP 1	MCW	LABELC-1	LABELC	7	Off sets constant (LABELC)
STEP 2	MCW	LABELA	LABELB	7	Inserts B-Op in constant.
STEP 3	MCW	LABELC	LABELA	7	Inserts N-Op in branch-out (LABELA)
LABELA	B	NSI		4	Branch out of subroutine.
	B	SUBROU		4	Branch back to subroutine.
		Total Positions:		29	
LABELB	NOP			1	
LABELC	DC	*NNN etc.		n	Any number of "N's" 1 or over.

NOTE: In addition to the instructions, you must consider LABEL B (1 position) and at least one "N" required for the constant (LABEL C). Actual total for this subroutine will then be 29 plus 1 plus n (at least 31).

Figure B-10 Rotary Switch Iteration Control Subroutine.

Figure B-11 illustrates the status of LABELA, LABELB and LABELC

PASS #	STEP #	LABELA Op	LABEL B CONTS	LABELC CONTENTS
1	1	B	N	NNN
	2	B	B	NNN
		N	B	NNN
2	1	N	B	BNN
	2	N	N	BNN
	3	N	N	BNN
3	1	N	N	NBN
	2	N	N	NBN
	3	N	N	NBN
4	1	N	N	NNB
	2	N	N	NNB
	3	B	N	NNB
5	1	B	N	NNN
etc.	2	B	B	NNN

Figure B-11Status of LABELA, LABELB and LABELC in the Rotary Switch Method of Iteration Control.

The subroutine would have branched to the address indicated here by NSI after pass 4, and would not have branched again until after 4 passes through thr routine, once this routine was entered.

This method of iteration control can be used for any type of count. It can be expanded to use any base. A DC (LABELC) of 2 N's will provide a count of 3, while a DC of 7 N's will provide a count of 8 iterations.

(B-12) Iteration-Counter Sign

1. Place a negative amount in the counter, and add one each time the routine is performed. When the counter reaches zero, test for a minus zero ($\bar{0}$) by using the Branch If Character Equal instruction -- B (III) (BBB) $\bar{0}$.

2. Same situation, but look for the sign change to plus by using branch if word mark or zone, BWZ (III) (BBB) B. This method may be used for any sized counter.

3. A counter can be worked in reverse by using a positive amount in the counter, subtracting 1 and testing for ($\bar{0}$). In the second situation, look for the sign change to minus.

4. A single position counter can be used for a count of 19 or less. For example, starting with -9 in the counter and testing for +9 (I) with the branch if character equal instruction.

(B-13) Programmed Word Mark Switches

Programmed switches can be an aid to subroutine selection. Basically, a programmed switch is a function that can be conditioned at one point in a program, to cause selection of alternate paths later on in the program.

Three steps in use of switch are:

1. Turning the switch on - SW 081
2. Testing the switch - BWZ III, 081, 1
3. Turning the switch off - CW 081

One method of effecting programmed switching involves the use of word marks (Figure B-12).

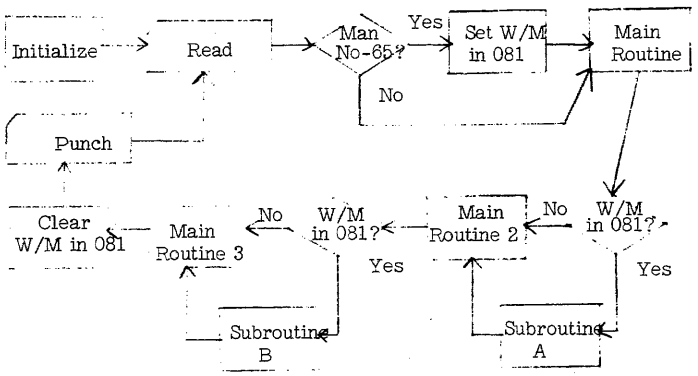


Figure B-12
Programmed Switching, Using
Word Marks

```
081       DC     #1
TEST      BWZ    III, 081, 1
TURNON    SW     081
          ---
TURNOF    CW     081          17 positions
```


(B-14) Programmed Character Switches

a. Branch if Character Equal

   1. One core location can be used for more than one "switch" condition. Thus, if one condition is met, set a WM. If another condition is met, insert yet another character, etc. The core position can then denote presence of a name and address card, (WM), and presence of YTD card (particular character, say a Y).

   2. A core position can store any of the many zone and/or WM conditions. These can then be tested with the Branch if Character Equal and Branch if Word Mark or Zone instructions.

b. Insert another Op-code to modify the program. A subroutine might change a NO Op to a Branch Op, then a N OP. (Figure B-13).

c. Change the d-modifier. Cause a branch on another sense switch, to another carriage tape channel, or branch on another I/O error condition by altering the structure of the d-modifier.

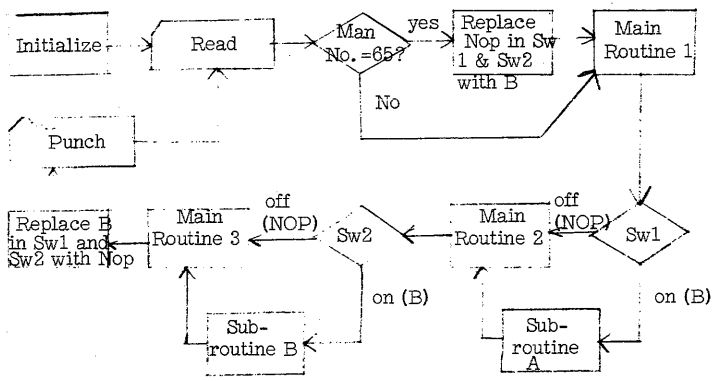


Figure B-13
Programmed Switching, Using
Op Code Alternation

```
SW1       B        On
          ---
TURNOF    MCW      @N@, SW1
TURNON    MCW      @B@, SW1          20 positions
```


(B-15) Branching Switches

1. The BRANCH IF INDICATOR ON instruction (B III d) may be used effectively for program switching by using an unused d-modifier (B III ¤ for example). To set the switch, simply set a word mark under the "¤" d-modifier. This makes the instruction an unconditional branch. To reset the switch, clear the word mark from under the d-modifier. Since the 1401 has no ¤ branch d-modifier, the instruction will cause no action, and become effectively a no operation instruction. The program will continue to the next sequential instruction.

```
Label     Op       Operand
SW1       BIN      ON, ¤
          --       --
          SW       SW1 + 4
          --       --
          CW       SW 1 + 4
```

13 positions

This switching method takes four less core storage positions than the conventional method of testing for the presence of a word mark.

CAUTION

If machine specifications are altered, what was once an unused d-modifier, may become used. This requires a review of all cases where this switch technique was used.

2. A similar approach would be:

```
NOP
B   XXX
```

OFF (N B XXX) ON (N B XXX)

By removing or placing a word mark under the B makes this a five position NOP instruction or (a one position NOP followed by) an unconditional branch. The same amount of memory is saved, as above; however, it is slightly slower in execution time but safer from the standpoint of changing d-modifiers.

3. Another way to program the branch switch would be:

Label	Op	Operand
TURN ON	SW	SW 1 + 4
	--	
SW1	BIN	TURN OF, ¤
	--	
TURN OF	MCE	

10 positions

The WM on the d-modifier in the branch instruction will be cleared by the edit instruction, if it is the first instruction in the "TURNOF" routine.

(B-16) Table Look-Up Programming

The object of table look-up is to find, from a group of table arguments, an item that equals (or in some cases approximates) the known search argument. The table functions (associated data) of table arguments can be located attendant to the argument, or a fixed number of positions away from the actual argument. Modifying the address of the location of the table argument will result in obtaining the location of its related function. Generally, some effort will be made prior to the actual search to determine whether or not the search argument lies within the limits of the table arguments.

Tables take many forms. They may be arranged sequentially in ascending or descending order. They may be arranged non-sequentially, with the high activity items appearing first in order by activity) followed by items with less activity. They may be arranged in random order, with access to the table arguments based on an address included in the input media. The nature of the application involved (equal hit, interpolation, etc.) will dictate the table design. In any event, table searching can readily be programmed on most data processing systems.

(B-17) True Binary Table Search

Table look-up becomes very time-consuming when directed at large tables or when long functions must be interspersed between the table arguments. No other programmed table look-up can completely search an ordered table in as few searches as the binary search.

The true binary search described here was developed for a situation where it was impossible to include the word marks needed by the table look-up instruction. It will completely search any size sequential table or group of records in a minimum number of comparisons, but does not require a great deal of storage for the program itself.

Theory of Binary Searching

Using a table that is either in ascending or descending sequence, it is possible to compare a search argument against the center table argument. If they are equal, the search is already finished. Otherwise the result of the comparison tells in which half of the table the desired argument may be found. A second comparison at the center on one of the halves can further tell which quarter of the original table might contain it. This procedure can be repeated as long as it is possible to subdivide whatever portion of the table remains.

Obviously, a table that can be repeatedly divided in half until only one logical entry is left must in itself be related to some power of 2. It must in fact contain a total of entries equal to one less than some power of 2 in order to simulate a "look-up equal" operation and exactly some power of 2 for "look-up equal-high" or "look-up equal-low".

The following table illustrates the application of a binary table:

Position in Table	Argument	Function
1	015	9463001
2	027	1004076
3	066	3472300
4	094	6875679
5	123	4221842
6	148	3884468

Position in Table	Argument	Function
7	159	5123779
8	177	6897212
9	200	2011897
10	251	3675774
11	283	2001480
12	694	7581531
13	733	0175000
14	746	6361792
15	999	*******

The table consists of 15 entries in ascending sequence. Each entry contains ten digits; three for the argument which is the field against which we must make our comparisons and seven for the function. The other element is the search argument which must match exactly with one of the table arguments.

To search this table by the binary method, the program must compare the search argument against the table argument of the eighth entry. If an equal condition results, the desired item has been found and the search terminates. However, if the search argument is low, the item must be among entries 1-7 of the table, and if high, it has to be in the upper seven entries (9-15). The next comparison is made on the item which is the next lower power of two entries away from the previously compared item. Thus, we must look at entry 4 (low) or entry 12 (high). Successive comparisons are then made which always reduce the number of possibilities by half until only one entry remains. If that entry is not equal to the search argument, it means that the argument is not in the table.

The course taken by the search is best demonstrated by using an actual input argument such as 123. This is initially compared against item #8 (177) and found to be low. The next comparison against the center item of the lower half of the table, item #4 (094), results in a high condition causing the search to move upward to item #6 (148). The low indication at this point means that only one possibility remains, item #5 (123), which in this case satisfies the equal condition desired.

To completely search a 15 item table such as the one on the previous page requires a maximum of only four comparisons. Note that the average number of comparisons is somewhat below this because if an equal condition results at some earlier point, no further comparisons are necessary.

As a binary table increases in size to one item less than higher powers of 2, the number of comparisons needed for a complete search becomes lower in relation to the size of the table. Thus the numbers of iterations needed for various larger tables are:

No. of Items In Table	Maximum No. of Comparisons
31	5
63	6
127	7
255	8
511	9
1023	10

The value of the binary table lies in the fact that it is perfectly symmetrical. Each successive comparison must move to a point higher or lower on the table which is exactly half the distance traveled by the previous comparison. When this distance has been reduced to the length of one item, the search is complete.

This type of table organization lends itself to computer programming in that a simple loop containing just one compare instruction, one of whose addresses is continually modified, can perform the whole search. A small subsidiary table contains values for each iteration equal to table address compared against at that point of the search. It must terminate with some indicator which tells the program that the last iteration has been completed.

For the 15-item table described, the subsidiary table would look like this:

```
4 X L = 40
2 X L = 20
1 X L = 10
End   = **
```


L equals the length of the table argument, plus the functions. This totals 10 digits in the previous example. After the initial comparison has been made at the table's center, the value 40 is added to the compare address if the result were high (subtracted if low). In this manner the programmed loop can accomplish the search. The program is controlled by the subsidiary table. By changing its values, the same program can operate on tables with different sized entries or, by adding more values at the beginning (80, 160, etc.), upon larger binary tables.

Any sequential table of any size can be made up of two overlapping binary tables of the next lower power of two minus one. Consider the following table of 25 items (an extension of the binary table of 15 items):

	Position In Table	Argument	Function
	1	015	9463001
	2	027	1004076
	3	066	etc.
	4	094	
	5	123	
	6	148	
Lower	7	159	
Binary	8	177	
Table	9	200	
1-15	10	251	
	11	283	
	12	694	
	13	733	
	14	746	
	15	758	
	16	762	
Upper	17	795	
Binary	18	796	
Table	19	811	
11-25	20	853	
	21	866	
	22	904	
	23	913	
	24	957	
	25	999	

It can be seen that this table may be thought of as one 15-item binary table of entries 1-15 and a second one consisting of entries 11-25. The only difference between searching this table and a straight binary table is that an initial comparison at the table's center (item #13 in this case) must force the program to search either the upper or lower table. The identical routine described can successfully search this unsymmetrical table. The only change is in the subsidiary table which controls the operation. Here one additional value must be placed at the beginning which will modify the address at item #13 to go next to either items #8 or #18 (the center points of the two binary sub-tables). The subsidiary control table would then appear as:

5 X L = 50

4 X L = 40

2 X L = 20

1 X L = 10

End = **

This method is valid for an equal search through any table having an odd number of entries. To handle an even number of entries requires a slight change because the initial distances moved (up or down) after the first comparison would not be the same. This is accomplished by creating two subsidiary tables instead of one. The increment table is referred to if the result of a comparison is high, the decrement table if low. If the original table were increased to 26 items, the subsidiary tables would appear as follows:

Increment Table	Decrement Table
60	50
40	40
20	20
10	10
**	**

The initial comparison could still be made against item #13, but if the result were high, the next comparison should be made against item #19 which is now the center of the binary sub-table extending from

item #12 to #26. Having the two subsidiary tables forces this sequence of operations. Note that this example requires a maximum of five comparisons, the number equal to the exponent of the next power of 2 which is greater than the number of items in the table.

Exactly the same routines can search for an equal argument in any size of table by adding more values to the subsidiary table (s). In this manner a table of as much as 2000 entries, for example, may be completely searched by comparing against only eleven (or less) of these entries.

(B-18) Programming Example of Binary Table Search for Equals

Care should be taken if the total length of the table exceeds 999 characters, that module 16 complements are used for the negative values in the subsidiary table: LOTBL. If the table exceeds 1999 characters, the constant: MIDPT, and possibly some of the positive values: HITBL, must be given their three-position equivalents. The program steps need not be changed.

Figure B-14 represents a logic diagram of a binary table search.

Figure B-15 illustrates the basic autocoder statements required for a binary table search.

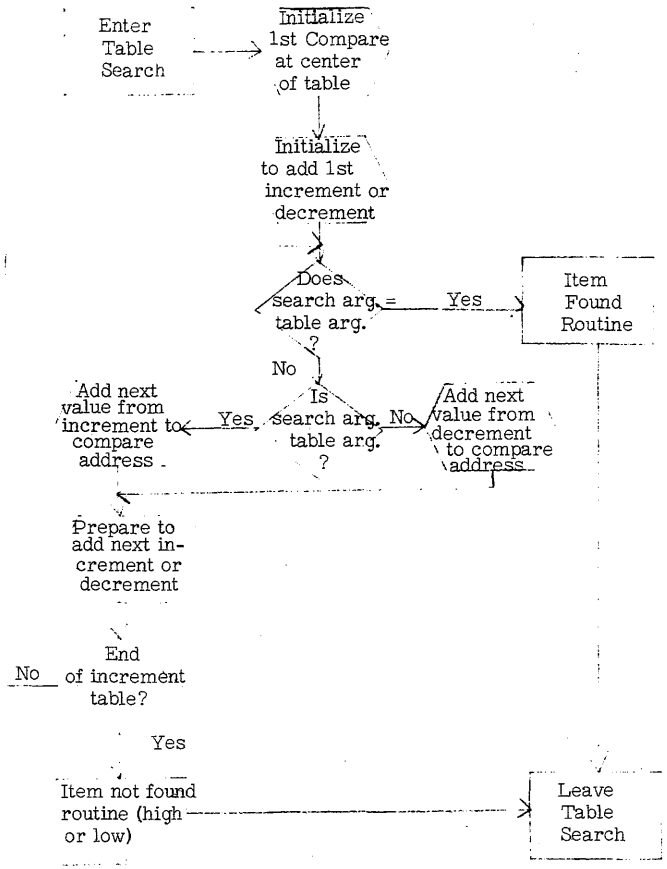


Figure B-14
Logic Diagram of Binary Table Search


```
* Binary Search Programming Example

START   ZA    MIDPT, X1       Initialize to middle item in
                              table
        ZA    &O, X2          Zero X2
COMP    C     TBARG&X1,       Compare Search argument
              INARG           to table
        BH    UPPER           Branch to go higher in table
        BE    FOUND           Branch if exact match
        A     LOTBL &X2, X1   Go lower in table
        B     ADDX2
UPPER   A     HITBL&X2, X1    Go higher in table
ADDX2   A     &3, X2          UP X2 for next value in
                              subsidiary tables
        BW    COMP, HITBL     Test for end of subsidiary
              -2&X2           table
NOFIND  ----  If branch on word mark not taken, item was
              not found.
FOUND   ----  Begin processing found item at this point.

*  Whenever an equal argument has been found, index register 1
*  contains the high-order relative address of the found table item
*  which may be processed as required.
*
*  Data areas needed

TABLE   DA    10X26           Table area of 26 items of 10
                              char. each
TBARG         1, 3            Table argument
TFUNC         7, 10           Table function
INPUT   DA    1X80, G         Sample input area
INARG         17, 19          Search argument

*  Subsidiary tables to control an equal only search of a 26 item
*  table containing 3 digit arguments and 7 digit functions.
*

LOTBL   DCW   -050            5 items lower in table
              -040            4
              -020            2
              -010            1
HITBL   DCW   &060            6 items higher
```

Figure B-15
Autocoder Program Segment of Binary Table Search

```
                &040            4
                &020            2
                &010            1
        DC      @ @     Lack of word mark here terminates
                        search.

*  Constant to initialize X1 to middle item of table.

MIDPT   DCW     &120
```

Figure B-15 (cont'd.)
Autocoder Program Segment of Binary
Table Search

(B-19) Binary Search for Equal-High

To simulate a "look-up equal-high" operation, only a small change in the interpretation of the table is required. The principal difference is that the ideal size table for this operation exactly equals some power of 2, rather than containing one item less. Recall that the chief virtue of the binary table is the fact that it is perfectly symmetrical for purposes of the search. Note that there is a subtle difference between an equal look-up and one for equal-high.

In the search for equal, each comparison eliminates one possibility, i.e., the item just compared. The two remaining halves of the table or sub-table must be of equal lengths. For this reason the comparison at the center point of the table or a sub-table must be at the center of a group that is one less item than some power of 2.

When the search is for equal or high, the item just compared is not necessarily eliminated from the search. A low result does not indicate that the desired table argument has been found until further comparisons have proven that the next lower item in the table is lower than the search argument.

Because the subsequent comparisons in the lower or upper halves of the table must take the identical course, and the item just compared must still be taken into consideration as an entry in the lower half, the table itself must contain a number of items exactly equal to some power of 2.

Although this difference exists, the logic of the search is unchanged. The only thing that must be altered is the points in the table where comparisons are made. This entails placing different values in the subsidiary control tables. The example of a 26-entry table (discussed in the previous section) was thought of as two overlapping binary tables of 15 items each (items 1-15 and 12-26) (for an equal search). To perform a "look-up equal-high" requires it to be thought of as two tables of 16 items each (items 1-16 and 11-26). After a comparison against the center item resulting in the search argument being found low, the next comparison would be made against item #8 in either case. However, if the result were high, the equal search would next look at item #19 whereas the equal-high search would be to item #18.

The logic and programming are almost identical for both an equal and equal-high search. The only difference is that the test for equal must branch to what was previously the "not found" routine. "Found" and "not found" are, therefore, synonymous for anything but an equal search. The reason for this is that there is no such thing as a "not found" condition following an equal-high search. The program must always find something, which generally means that the last item in the table should be a pad of 9's or some other unique indicator.

The subsidiary tables to control the equal-high search on the same 26 item table would appear like this:

```
LOTBL    DCW    - 050    HITBL    DCW    + 050
                - 040             + 040
                - 020             + 020
                - 010             + 010
                - 000             + 010
                                  @ @
```

Note that these tables each contain one more entry than the corresponding tables for an equal search. If, on the last iteration of an equal-high search, the search argument is found to be low, the proper "higher than" item is the one just compared. If the search argument is high, the next higher item in the table is the one desired. Therefore, after the final comparison, a low result leaves the table argument's address unchanged (LOTBL entry -000), but a high result causes it to be adjusted upward by one item (HITBL entry +010).

If a search for an argument just lower than the highest item in the table is followed through, this final table argument is never actually compared in the example given. It is assumed to be the desired item if the search argument is greater than the next-to-last item. For this reason, it is best to tag this last item with some special indicator (such as 9's), which can be identified by the program.

The search for equal-high takes the same maximum number of comparisons as the search for equal. However, in many applications the equal condition (the only condition that can stop the iterations before maximum) occurs only rarely while looking for equal-high. Here the average number of comparisons per search would be nearly the same as the maximum.

(B-20) Binary Search for Equal-Low

A slightly different situation exists when the search requires a table argument that is either equal to, or the next lower value than, the search argument. Again, the subsidiary control tables must be altered, with no change in the program instructions.

The difference is that comparisons to every level of a binary table or sub-table are made against the left-of-center item for an equal-high search and against the right-of-center item during an equal-low search. If it is lower (search argument high), the desired item belongs to the upper half. The situation is reversed in a search for equal-low where the significant point of comparison is at the lowest item in the upper half of the table segment. When the search argument is low or equal, the search continues in the upper half. If high, the search must shift down to the lower half. This reasoning is valid if the table contains entries totaling any power of 2 (or 2 itself, where the comparison pinpoints the proper item, which is what happens on the final iteration of the sample program).

To cause the same sample program to follow the desired sequence of comparisons for the equal-low search on the 26-item table, the following subsidiary control tables would be employed.

```
LOTBL    DCW    - 040    HITBL    DCW    + 060
                - 040             + 040
                - 020             + 020
                - 010             + 010
                - 010             + 000
                                  @ @
```

In this example, the initial comparison is still being made against item #13 (MIDPT equals +120). The point of initial comparison is not fixed on one particular item, but on any of those that are part of the overlapping portion of the two binary sub-tables, provided the first increments in the subsidiary tables are adjusted accordingly. Thus, if MIDPT were made +130 (to compare first against item #14), the first entries in the two subsidiary tables would be -050 and +050 (for a 26-item table).

The equal-low search requires a special test character in the lowest (left-most) position of the table. The program arrives at this entry without comparing against it if all other items have been found to be higher than the search argument. It can contain asterisks or some other indication that may be tested by the program.

Timing for an equal-low search is the same as for equal-high.

(B-21) Binary Search for Tables in Descending Sequence

Tables arranged in a descending sequence may be searched by the same program by changing the points of comparison so that they are oriented toward the right end of the table in the manner that the ascending table's points are oriented toward the left end. When the result of any one comparison indicates that the next point should be higher in the table (at some higher table argument), the address of this point is arrived at by decrementing the current address. This means that if the initial point of comparison is moved one item to the right, just by changing the signs on the values in the increment and decrement subsidiary tables, a search of an ascending table can apply to the same size descending table.

To continue with the sample table of 26 items used in previous illustrations, the three types of look-up when applied to descending tables would require the following subsidiary tables (MIDPT is +130 or item #14).

Look-Up Equal

```
LOTBL    DCW     + 050    HITBL    DCW     -060
                 + 040                     - 040
                 + 020                     - 020
                 + 010                     - 010
                                   DC      @ @
```

Look-Up Equal-High

```
LOTBL    DCW     + 050    HITBL    DCW     - 050
                 + 040                     - 040
                 + 020                     - 020
                 + 010                     - 010
                 + 000                     - 010
                                   DC      @ @
```

Look-Up Equal-Low

```
LOTBL    DCW     + 040    HITBL    DCW     - 060
                 + 040                     - 040
                 + 020                     - 020
                 + 010                     - 010
                 + 010                     - 000
                                   DC      @ @
```

(B-22) Construction of Binary-Search Subsidiary Tables

Calculation of binary-search subsidiary tables is to be based on the following:

- Select initial point of comparison
- Determine the first value in both the increment and decrement tables
- Fill in the remaining values except the last
- Select the last value.

The initial point of comparison is generally at the mid-point of any table, although it may be against any of those items which fall in the overlapping portion of the two binary sub-tables of the next lower power of 2. Let us call this entry number M. From this, the value of the constant, MIDPT, can be calculated relative to zero. Where L = the length of each table entry:

$$MIDPT = L(M-1)$$

The second points of comparison are the most critical because they are peculiar to the type of search performed. For an equal-high search, T, the total number of items in the table must include the highest possible argument (or a special indicator). The same is true of the lowest argument in an equal-low search. Compute the value of n (the power of 2 that is the next lower to the total items) so that:

$$2^n < T < 2^n + 1$$

Having determined the values of M, L, T and n, find the items that may be compared against on the second iteration. Consider tables in ascending sequence. The following formula table (Figure B-16) shows how the two points are arrived at for the different types of search:

Type of Search	Item # After Low Result (P_1)	Item # After High Result (P_2)
Equal	2^{n-1}	$T - 2^{n-1} + 1$
Equal-High	2^{n-1}	$T - 2^{n-1}$
Equal-Low	$2^{n-1} + 1$	$T - 2^{n-1} + 1$

Figure B-16
Second Points of Comparison for Ascending Tables

The above points may be called P. Thus, for example, if a table contains 53 items for an equal-low search, T=53 and n=5 (i.e., 2^5 53 2^6). When the result of the initial comparison is low, the second should be made at item P_1 where:

$$P_1 = 2^{5-1} + 1$$

$$P_1 = 17$$

If the result is high, it must be made at item #38.

$$P_2 = 53 - 2^{5-1} + 1$$

$$P_2 = 38$$

To obtain V_1, the value to be placed in the decrement table (LOTBL), the V_2, to be the first entry in the increment table (HITBL) merely subtract the value of M from the respective P and multiply by the length of each table item. Thus:

$$V_1 = L(P_1 - M)$$

$$V_2 = L(P_2 - M)$$

The same type of formula table for descending tables applies (Figure B-17).

Type of Search	Item # After Low Result (P_1)	Item # After High Result (P_2)
Equal	$T - 2^{n-1} + 1$	2^{n-1}
Equal-High	$T - 2^{n-1} + 1$	$2^{n-1} + 1$
Equal-Low	$T - 2^{n-1}$	2^{n-1}

Figure B-17
Second Points of Comparison for Descending Tables

The values of V_1 and V_2 are determined by the same equations shown above.

It can be seen that the values of V_1 and V_2 are functions of the number of items in the table, the length of each item, the sequence of the table, the type of search to be performed, and the position of the initial comparison.

The assignment of specific values to V_1 and V_2 forces the search's second iteration to look at the center of some size binary table. For an equal-only search, this is exactly at the center item. When the look-up is for equal-high, it is the highest item of the binary table's lower half; for equal-low, the lowest item of the upper half.

After the second comparison, the search assumes a regular pattern for every type of look-up and previous result. Therefore, these portions of the two subsidiary tables are identical except for the signs of the values. They follow this progression:

$$L\,(2^{n-2}),\quad L\,(2^{n-3}),\quad \ldots\ldots L\,(2^{0})$$

When dealing with an ascending table, all signs in the decrement (LOTBL) table are minus and in the increment (HITBL) table, plus. The signs are reversed for a descending table. Note that this portion of a subsidiary table always contains n-2 values.

At this point the subsidiary tables are complete for an equal-only search where, in the event of a not-found condition, the programmer is not concerned with the next higher or lower items. All that must be added is the "search-end indicator" which, in the sample program, is the position without a word mark (DC) following the increment table (HITBL).

For an equal-high or equal-low look-up the final element of a subsidiary table is added to the address of the very last item compared causing the program to "find" it, or the item immediately to the right or left. If the item in the final comparison is the one desired, this element is zero. Otherwise it is plus or minus the item length (L) causing the search to end one item higher or lower in the table. This value is constant as follows for the different table sequences and types of search. (Figure B-18).

Type of Search	Decrement (LOTBL)	Increment HITBL)
Ascending Tables		
Equal	Not required *	Not required *
Equal-High	Zero	+L
Equal-Low	-L	Zero
Descending Tables		
Equal	Not required *	Not required *
Equal-High	Zero	-L
Equal-Low	+L	Zero

Figure B-18
Final Subsidiary Table Values

* Note

By including the same values required by equal-high or equal-low searches, a not-found condition following an equal search can pinpoint the next higher or lower item respectively.

The final increment table item must be followed by a location not containing a word mark to stop the iterations.

By following the above steps the user can easily adapt the sample standard search program to perform any of the usual table look-up operations upon any sequential table containing elements of a fixed length. Since the size of the area being searched is controlled by the size of the subsidiary tables, the latter may be modified by a program (such as an internal sort) to expand as the table (addresses of already sorted records) increases.

Conclusion - True Binary Table Search

The true binary search has wide application for any computer not equipped with table look-up instructions. Some alternative method might be preferable when dealing with tables containing only a few items, or if the frequency of "hits" is disproportionately large on a very small number of items.

The storage requirements of the program and subsidiary tables are only slightly greater than for the simplest type of indexed search. No other programmed table look-up can completely search an ordered table or group of records in as few iterations as the binary search. The number of steps actually executed per iteration is hardly greater than by the usual, less efficient routines.

The binary search, of course, cannot operate upon non-sequential tables.

(B-23) Direct Address Table Searching

The table functions are arranged so that the coded address of any of them is included in the input media. Direct reference to this address locates the data required. The tables entries may be in random order.

Example:

A table containing 90 items (10 characters per item) is arranged so that the actual address of each item in the table is coded in the input media (card). The table is located in storage locations 091 to 990. The table function addresses are punched in column 1, 2 and 3 and range between 100 and 990.

To find the table functions that relate to the coded input card, compare the input records to its associated table function. If the item is not in the table, this fact is indicated. (See Figure B-19)

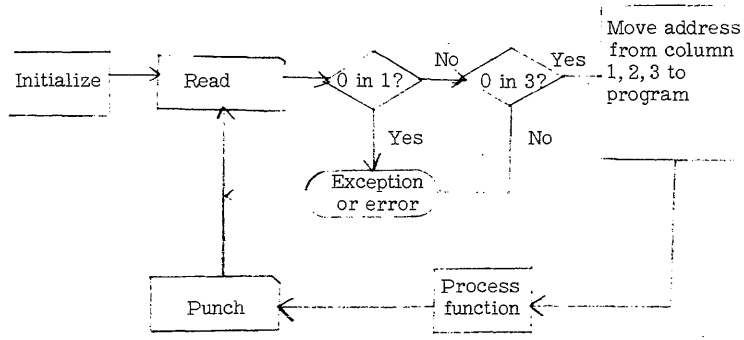


Figure B-19
Direct Address Table Look-Up Search

(B-24) Successive Table Searching

This method finds its greatest value in instances of short tables with high activity in the initial items. It consists simply of beginning with the first item in the table and comparing each of these table arguments in succession, with the search argument. Example: A short (20 item, 5 characters per item) table is arranged in random order, with the items with the highest activity first. The table functions are located with, and immediately to the left of, each of the table arguments.

The table argument in storage that equals the search argument in the card can be found, as shown in Figure B-20. Each card is processed on the basis of its data in relation to each successive entry in the table. If the item is not in the table, this fact will be indicated. Note that the use of indexing and address modification would greatly facilitate this search.

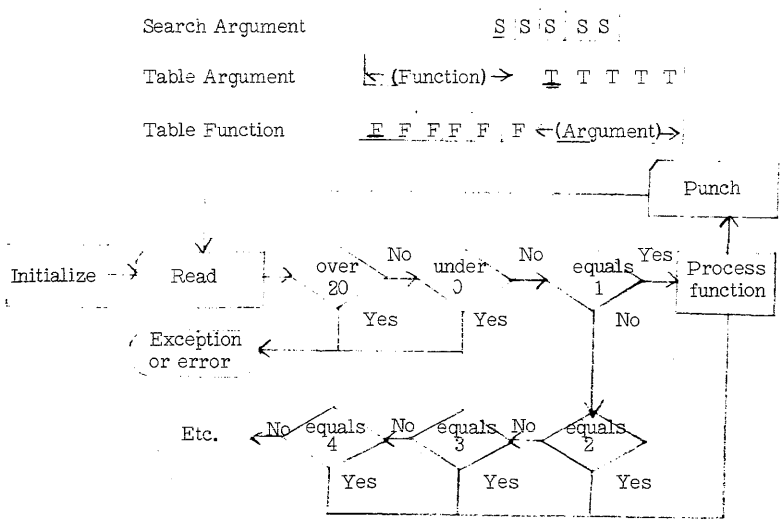


Figure B-20
Successive Table Look-Up Search

(B-25) Special Table Searching

There may be instances where information in the input media will tend to point directly to specific table areas, precluding the necessity of a tedious general search. The programmer should be alert to these possibilities.

Example:

A table containing items is arranged in six 10-item sections (10 characters per section). A 5-item section-table of two characters per item serves as a locator. The section number is punched in columns 78 and 79 of the input card. Column 80 indicates the item number within the section. Thus, item number 3 within section AB is defined as AB 3.

To find the table function in storage associated with the code punched in the card, process each card based on the stored data and that already punched in the card. If the item is not in the table, this fact will be indicated (See Figure B-21). Note that this method is basically a combination of the successive search and the direct addressing search methods.

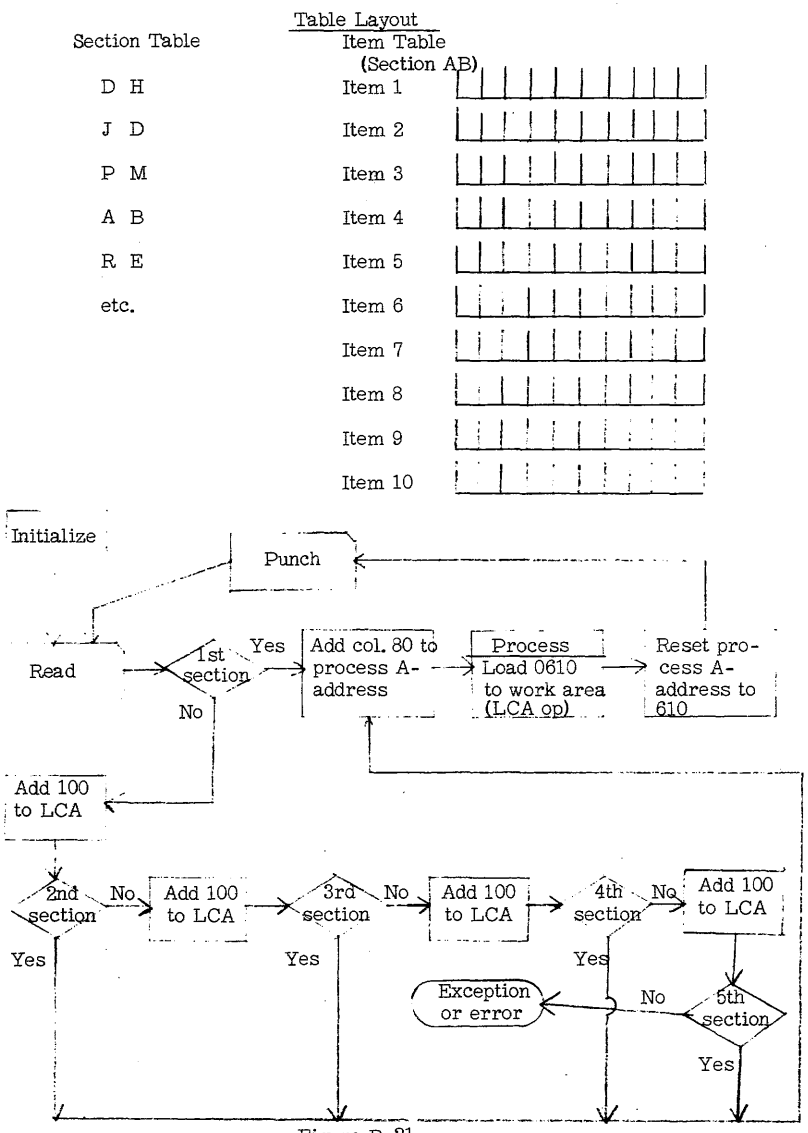


Figure B-21
Special(Combination) Table Look-Up Search

(B-26) Table Search by Bracketing

This method of table search is utilized primarily in those instances where the table is made up of relatively equal activity items. This technique differs from binary table search. In this approach (Figure B-22):

1. The search argument is compared with the middle table argument. If this table argument is equal to the search argument, the search is complete.

2. If these two arguments are not equal, the search argument is then compared to a table argument that brackets the next set of possibilities in the upper or lower part of the table. (The decision to search further in either the upper or lower part of the table is based on the relationship of the original comparison and the construction of the table.)

3. This process is repeated until a "hit" (equal or acceptable approximate situation) is obtained.

Example:

A table of several items (2 characters per item) is arranged in ascending order. Each of the items are equally active. The functions are located with and immediately to the left of the table arguments.

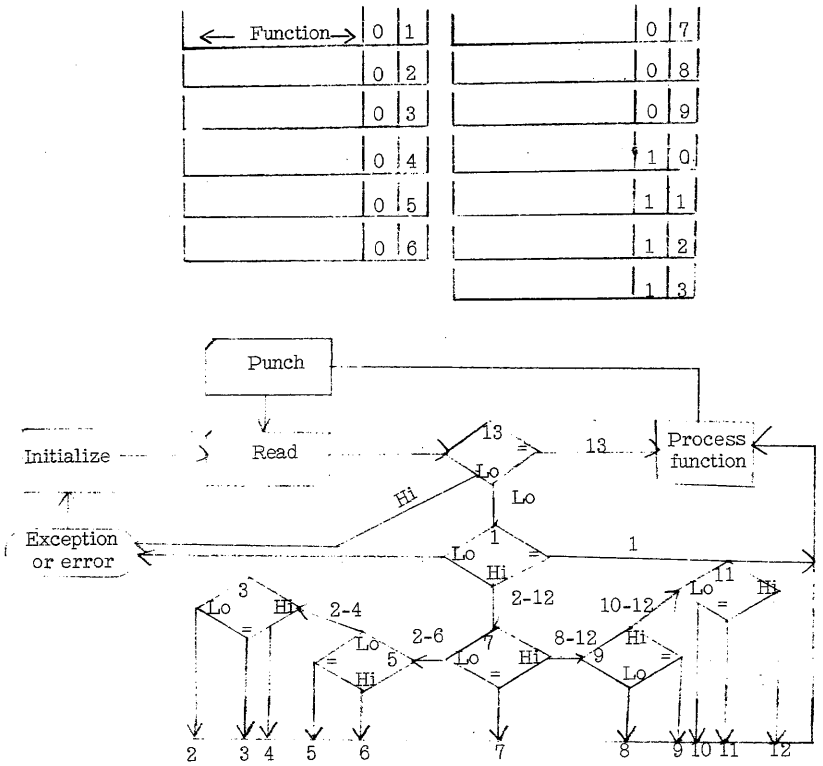


Figure B-22
Bracketed Table Look-Up Search

Note that this bracketing search method of table look-up would be greatly facilitated by the use of address modification and indexing.

(B-27) Clearing Storage Between Limits

1. An approach to clearing storage involves the use of the move record op.

```
CS   332
CS
MRCM  200, AREA
```

A G/M - W/M or R/M must be in location 333. The A-address of the MRCM instruction can be altered to clear an area of 1 to 133 locations.

2. If the move record op is not available, the following routine can be used.

```
CS    332
CS
SW    200
LCA   332, FIELD
CW    200
```

A W/M in 199 could be set in housekeeping if the location is not otherwise used. The SW and CW instructions would not then be needed. The A-address of the LCA instruction can be decreased to clear an area of 1 to 133 locations.

(B-27.1) Clearing to Zero

1. A fast way to clear index registers to true zero. Assuming there are word marks in locations 087, 092 and 097:

```
S     100
S
S
```

This places the signs generated from the subtraction in 100, 096 and 091, and the index registers are true zero.

(B-28) 80 Column Card Reproduce Routine

Figure B-23 shows a method of preparing a single card reproduce program for the 1401 or 1460.

The problems to be considered with the 1440 revolve around the method of reproduction and machine configuration. There are at least three approaches to reproducing on the 1440:

1. When only one 1442 is available without disk, it is necessary to merge blank cards behind the cards to be reproduced.

2. When two 1442's are available, the routine would read in on one 1442 and punch out on the other 1442.

3. If disk tracks are available, the input deck could be stacked on the file. Then, after all the cards are read, load blank cards into the 1442 and punch out from the disk. This takes a load program and a dump program.

The routines to accomplish the 1440 reproduce function are too long to be considered for inclusion in this manual.

Step No.	Card Column	Instruction OP	A/I d	B d	d	Remarks
1	1-7	,	0085	0156		Set w/m for step 2 and 3
2	8-14	,	0222	0226		Set w/m for step 4 and 5
3	15-21	L	0774	1122		Move program from card area to program area
4	22-25	B	361			Branch to program area
5	26-32	,	3683	7122		Set w/m for step 6 and 7 (in program area)
6	33-36	/	080			Clear read area
7	37-43	,	0013	7129		Set w/m in 001 and for step 8
8	44-50	,	3863	9133		Set w/m for step 9 and 10
9	51-57	,	4004	0101		Set w/m for step 11 and 12
10	58-64	,	4084	1122		Set w/m for step 13 and to end its execution
11	65	N				No op
12	66-72	L	0801	8000		Move read area to punch area
13	73-76	5	4004			Punch-read and branch to step 12

These instructions are punched in a single card and placed in front of the cards to be reproduced. The actual reproduce routine is only two steps (12 and 13). All the other steps merely clear core in the necessary areas and transfer the routine from the initial read in area to the program area.

Figure B-23

Single Card Reproduce Routine, 1401/1460

620405M5

(B-29) 80 Column Card Gang-Punch Routine

Figure B-24 shows a method of preparing a single card gang-punch program for the 1401 or 1460.

Gang punching on the 1440 would involve a program that would read the card to be gang punched and then punch the data into the cards that followed. More than one card would be needed to execute this program.

Step No.	Card Column	Instruction OP	A/I d	B d	d	Remarks	Effective No. of Characters Inst	Data	Total
1	1-7	,	0 0 8	0 1 5		Set w/m for step 2 and 3			
2	8-14	,	0 2 2	0 2 9		Set w/m for step 4 and 5			
3	15-21	L	0 8 0	4 8 0		Move program from card area to program area			
4	22-28	,	4 3 4	4 7 8		Set w/m for step 6 and to end program execution			
5	29-33	B	4 3 4			Unconditional branch to program area			
6	34-40	,	4 4 1	4 4 8		Set w/m for step 7 and 8 (in program area)			
7	41-47	,	4 5 5	4 5 9		Set w/m for step 9 and 10 (in program area)			
8	48-54	,	4 6 6	4 6 7		Set w/m for step 11 and 12 (in program area)			
9	55-58	/	0 8 0			Clear read area			
10	59-65	,	4 7 4	0 0 1		Set w/m for step 13 and in 001			
11	66	1				Read card to be gang-punched			
12	67-73	L	0 8 0	1 8 0		Move read area to punch area			
13	74-77	4	4 7 4			Punch and branch to step 13			
14	78					Final w/m location			
						These instructions are punched in a single card and placed in front of the card containing the data to be gang-punched. The actual gang-punch routine is steps 11-13. All other cards set up w/m's, clear core and transfer data. Blank cards should be placed in the punch feed. Data in the gang-punch master is punched column for column in the blank cards. If pre-punched cards are used in the punch feed, turn the I/O check stop switch off.			

Figure B-24 - Single Card Gang-Punch Routine, 1401/1460

620405MSF

(B-30) Relocatable 1401 Core Storage Print-Out Routine

1. Specify the beginning address desired in the A operand of the ORG card.

2. If assembled separately:

   a. Discard the first two cards produced by the 1401 SPS processor. These are a clear storage routine.

   b. Discard the last card produced by the 1401 SPS processor. This is a transfer card.

   c. Place this assembled print-out routine before the transfer card (the last card) of the program in which the print-out is to be included.

3. If assembled with your program:

   a. Punch the A operand of the END card with the proper start location of your program.

   b. Do not include an END card between your program and this routine.

4. Any time that you wish to do a storage print out, manually branch to the location that you have specified in the A operand of the ORG card. The contents of the Print area (with word marks) will be printed first, followed by locations 001 to 100, 101 to 200, etc. Each 100 character core strip will be identified by an upper and lower limit indication on the far right (print positions 301-332). The word mark associated with each core position appears as a 1 beneath it.

5. This routine has not disturbed your program, other than to have destroyed what was in the Print area (201-332). Restart at any point without reloading the program.

IBM

Program

Programmed by

Date

INTERNATIONAL BUSINESS MACHINES CORPORATION

IBM 1401 AND 1410 DATA PROCESSING SYSTEMS

AUTOCODER CODING SHEET

Form X24-1350-1
Printed in U.S.A.

Identification 76 80

Page No. 1 2 of

Line	Label	Operation	OPERAND	
0,1		Relocatable 1401 storage print-out subroutine		
0,2		Origin card not required if assembled with another program		
0,3		Discard clear-storage card if assembled separately		
0,4		This program uses the MA and SBR op codes. Notes are included to eliminate these.		
0,5				To eliminate SBR+MA op
0,6		ORG	3500 Start here	
0,7	PRCORE	CS	0 Establish storage capacity constant.	Create DCW giving
0,8		SBR	HICORE# 3 Store core capacity	core limit.
0,9		W	Print 201 thru 332	
1,0		CC	S Double space after next print	
1,1		W	# Print word marks.	
1,2		CS	332 Clear upper print area	
1,3		SBR	TRANS+6,,201 Setup transfer operation	Setup DCW's with 201 and
1,4		SBR	TRANS+3,,001 Setup transfer operation	001 and MCW to setup transfer
1,5		LCA	CONHI,,319 Load hi-limit indicator	
1,6		LCA	Load lo-limit indicator	
1,7		SBR	TEST+6,,001 Setup WM test	Setup DCW with 001 and
1,8		SBR	SET+3 Setup WM set	MCW to setup W/M tests.
1,9		SBR	CLRNOP+3 Setup set on clear Wm op	CW in TRANS+4 and
2,0				TRANS+1; label CLRAB
2,1	TEST	BWZ	YESWM,1,,1 Test WM in 0001	
2,2		MCW	@#@,,CLRNOP Set CLRNOP to clear WM	
2,3		B	SET Branch to set	
2,4	YESWM	MCW	@N@,,CLRNOP Set CLRNOP to NOP	
2,5	SET	SW	1 Set WM in low limit	

Figure B-25 - a. Re-locatable Storage Print-Out Routine (Autocoder)

620640MSP

B-62

IBM

Program

Programmed by

Date

INTERNATIONAL BUSINESS MACHINES CORPORATION

IBM 1401 AND 1410 DATA PROCESSING SYSTEMS

AUTOCODER CODING SHEET

Form X24-1350-1
Printed in U.S.A.

Identification 76 80

Page No. 1 2 of

Line	Label	Operation	OPERAND	
0,1	TRANS	LCA	1,,201 Transfer data to print	
0,2				SW in TRANS+1
0,3	CLRNOP	NOP	1,,201 Clear WM or NOP	
0,4		C	TRANS+4,,@3@ Comp for 3XX in TRANS B-add.	
0,5		BU	CMPTRA BR if not 3XX in TRANS B-add.	
0,6		W	Prt. if 3XX in TRANS B-add.	
0,7		CC	S Double Space	
0,8		W	# Print word marks	
0,9		MA	@000@TRANS+4 +1 TRANS A, -1 TRANS B	Change to A op
1,0		SW	SET+1 Modify set, test and clrnop to higher core area	
1,1		MA	@001@,SET+3	Change to A op
1,2		MCW	SET+3,TEST+6	
1,3		MCW	SET+3,CLRNOP+3	
1,4		CW	SET+1	
1,5		A	@001@,307 Increment lo-limit indic.	
1,6		A	@001@,,317 Increment hi-limit indic.	
1,7				SW in TRANS+4
1,8	SETB	MA	@001@,TRANS+6 Increment TRANS B-add.	Change to A op
1,9		B	TEST	Go to CLRAB
2,0	CMPTRA	C	TRANS+3,HICORE Comp. TRANS A-add to hi core	
2,1		BU	INCTRA Test if hi core	
2,2		CS	300 If not, clear 300	
2,3		MN	@9@,,319 Decrement hi limit indic. from hi to hi-1	
2,4		A	@I9I@,318	
2,5		W	Print data final	
		W	# Print W/M final	
		H	End of job	
		LTORG		

Figure B-25 b. Re-locatable Storage Print-Out Routine (Autocoder)

620640MSP

B-63

IBM

Form X24-1350-1
Printed in U.S.A.

Program _____

Programmed by _____

Date _____

INTERNATIONAL BUSINESS MACHINES CORPORATION

IBM 1401 AND 1410 DATA PROCESSING SYSTEMS

AUTOCODER CODING SHEET

Identification _____ (76 80)

Page No. _____ (1 2) of _____

Line (3 5)	Label (6 15)	Operation (16 20)	OPERAND (21 25 30 35 40 45 50 55 60 65 70)	
0 1				
0 2	INCTRA	MA	@0 0 1,@, TRAN,+3,	Increment TRANS A-add. Change to A op.
0 3		B	SETB	Branch to SET B
0 4	CONLO	DCW	@ 0 0 0 1 THRU @	
0 5	CONHI	DCW	@0100@	
0 6		END	PRCORE	
0 7				
0 8				
0 9				
1 0				
1 1				
1 2				
1 3				
1 4				
1 5				
1 6				
1 7				
1 8				
1 9				
2 0				
2 1				
2 2				
2 3				
2 4				
2 5				

Figure B-25 c. Re-locatable Storage Print-Out Routine (Autocoder)

620640MSP

(B-31) Field Inversion Routines

When it is necessary to invert an entire field within the same locations in core, one of the following techniques can be used. Figure B-26 shows the Revolving Method used in inverting a 5 position field. With an odd number of characters in the field, the middle character remains the same. Word marks in the original field are unchanged.

LABEL	OP	OPERAND	COMMENTS
CR	DCW	0	Location to save character
	MLC	FLD, CR	Save units position
	MLNS	FLD-4, FLD	Move digit portion of high order position to units
	MLZS	FLD-4, FLD	Move zone portion of high order position to units
	MLC	CR, FLD-4	Re-insert character in hi order position
	MLC	FLD-1, CR	Interchange
	MLNS	FLD-3, FLD-1	Second and
	MLZS	FLD-3, FLD-1	Fourth
	MLC	CR, FLD-3	Positions

Figure B-26
Field Inversion Routine Using Revolving Method

It can be seen that each pair of characters to be inverted requires a routine of 4, 7 position instructions. By changing the address increment, the routine can be altered to handle any size field inversion. If index registers are available, two can be used to increment and decrement the addresses of "FLD". It would pay to use index registers if the field to be inverted is extremely long. The technique illustrated above would require:

Field Size to be Inverted	Core Required
2-3	29 positions
4-5	57 "
6-7	85 "
8-9	113 "

Figure B-27 shows the Slide Method of field inversion (5 position field). This method must have a word mark in the high order position of the field to be inverted and no others. The routine can be altered to handle any field size by changing the d-character of the Branch if Character Equal instruction and including the proper number of clear W/M instructions.

LABEL	OP	OPERAND	COMMENTS
CTR	DCW	0	Counter
CR	DCW	0	Location to save character
INVERT	ZA	INVERT, CTR	Use +0 op code to reset counter
	MLC	FLD, CR	Save units position
	MLCWA	FLD-1, FLD	Slide field 1 position
	MLC	CR	Re-insert saved character
ADD	A	ADD, CTR	Use +1 op code to add 1 to counter
	BCE	OUT, CTR, D	Branch out after CTR reaches +4

Figure B-27
Field Inversion Routine using Slide Method

Figure B-27 cont'd.

LABEL	OP	OPERAND	COMMENTS
	B	INVERT + 7	Continue slide
OUT	CW	FLD, FLD-2	Clear W/M's from all but high order character
	CW		

The core requirements for this technique are:

Field Size to be Inverted	Core Required
2	50 positions
3	53 "
4	57 "
5	54 "
6	58 "
7	55 "
8	59 "
9	56 "

(B-32) Job Initialization Routines

Several functions usually must be completed before the body of the job can logically proceed. These include the setting of word marks in the card read in area, the initial setting of index registers and counters, such as page numbering counters, etc. This initialization subroutine should be programmed so as to allow a job-restart without loss of any of the program's usefulness. These routines are often referred to as housekeeping routines.

1. Where a job requires all available core storage, the initialization routine can be loaded and executed. The main program can then be loaded over the initialization instructions (overlayed).

2. If possible, use ORG to origin housekeeping routines in the punch or print areas, which can be cleared by CS OPS. This will not cost storage useable in the main program, and eliminates necessity for overlaying.

3. When a programmer uses EX to execute instructions, and then overlay the area, he must provide his own linkage back to the load routine. In the 1440, the reentry point is the 9th position of the loader.

(B-33) Storage Locations 000 and 100 (1401/1460)

Storage location 000 is used for an internal timing count when card reading is in progress. On any program step not relative to card read (op-code of 1, 3, 5, 7, or 4R, or 6R), this position may be used provided the instruction does not decrement the address 000. This core location will contain AB bits after a read op. This zone is present, but cannot be accessed because any op capable of moving it will decrement the address 000 to high order of core, and cause a wrap-around error. It may be used as the first character of a tape record, or a move record op (P), since these functions will cause the registers to increment.

Storage location 100 will contain a 0 (8 and 2 bits) after a punch op.

(C) CPU Operating Pointers, and Miscellaneous Error Indications

(C-1) Operator Control Console

1. The 1401 Mode Switch cannot be used to cycle through an input-output operation. When this switch is in the Single Cycle Process or Single Cycle Non-Process mode, all I-cycles will be taken one at a time, but all the B-cycles will be taken at processing speed.

2. The I/Ex mode may be used for any I/O op.

   CAUTION

   When operating the I/Ex mode and the program reaches a point where you wish to alter a character, turning the Mode Switch to "alter" requires passing the "run" position. Occasionally, the machine will start running without hitting the start key.

3. The Address Stop mode is not effective for any I/O op. If you want to stop processing at the address of an I/O op, address-stop on the preceeding op-code.

(C-2) Console Error Log Sheets

Some error-logging procedure should be provided, so that the operator can note the condition of the console indicators at the time of the error. This provides an aid to both the programmer and the Customer Engineer. This is especially helpful where the error occurs rarely, or only after the operation is well underway.

Figure C-1 represents one such console error-log for 1401 and 1460.

Figure C-2 represents such a log for the 1440.

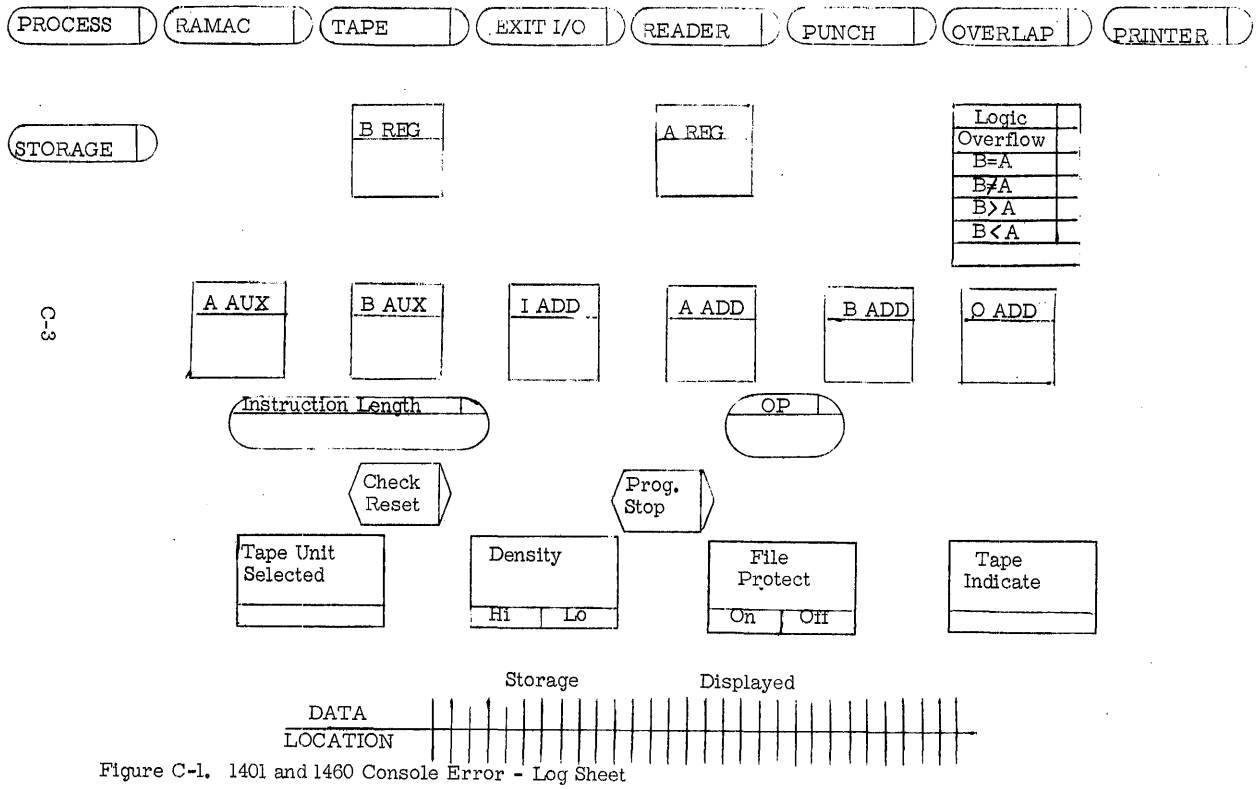
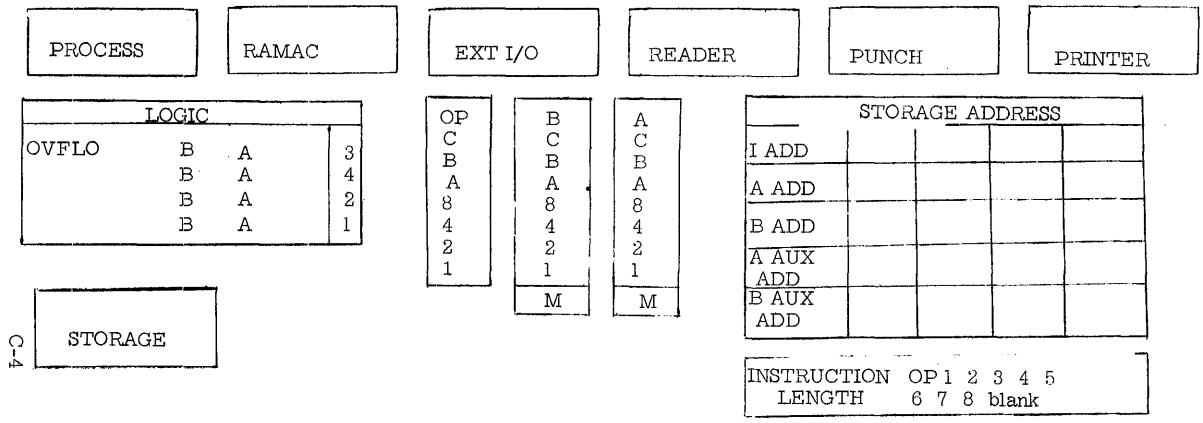


Figure C-1. 1401 and 1460 Console Error - Log Sheet

1440 CONSOLE DEBUGGING LOG

PROGRAM_____

TIME_____ DATE_____



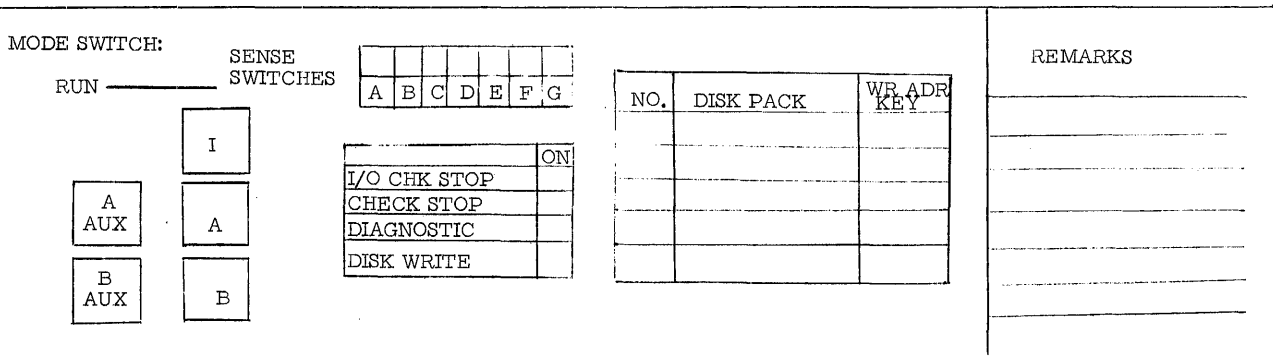


Figure C-2. 1440 Console Error-Log Sheet

(C-3) Program Analysis Charts

The information contained in the following chart, Figure C-3, will not only assist in checking out programs, but in differentiating between program errors and system malfunctions. A customer engineer may be needed to correct some of the malfunctions.

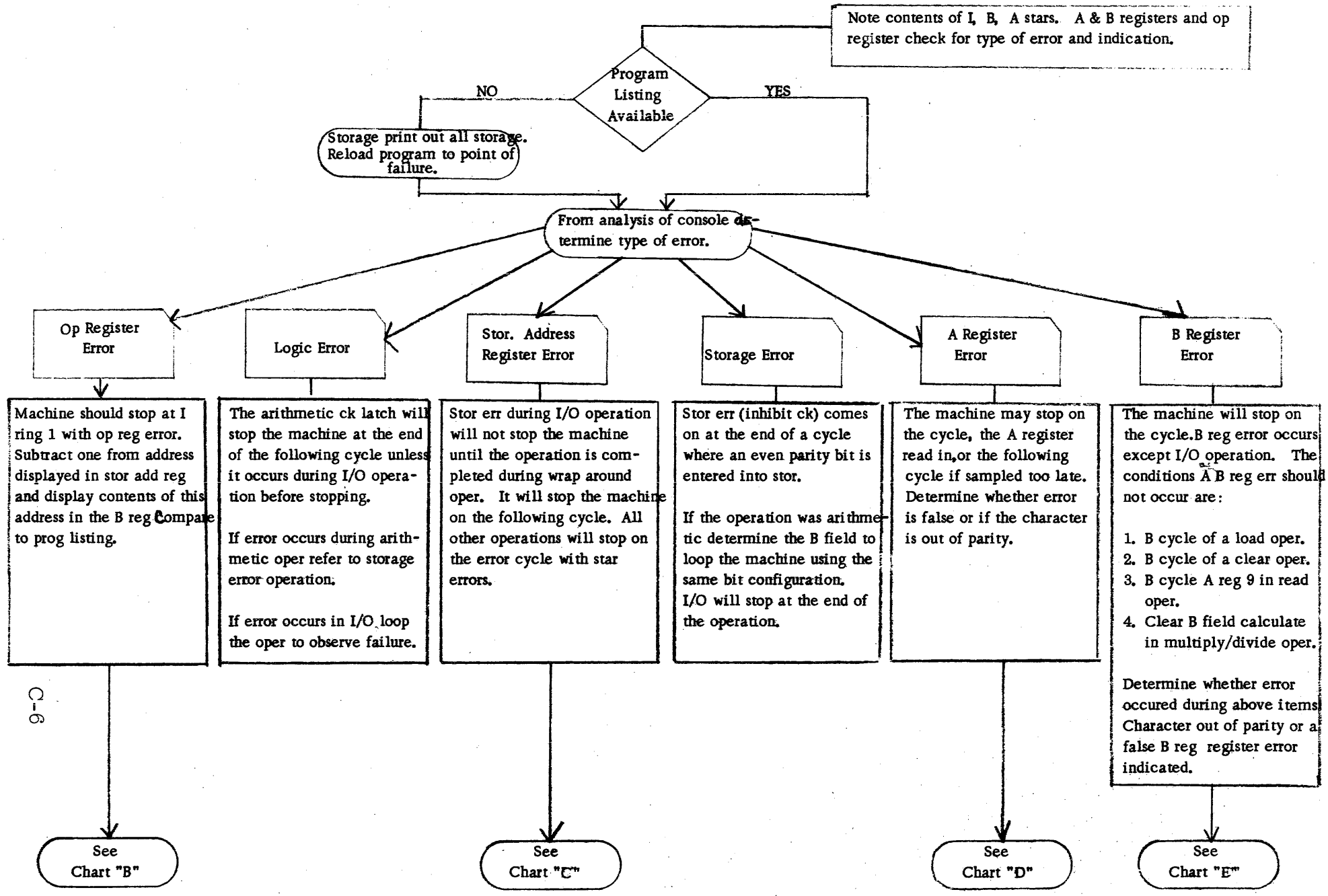


Figure C-3A
PROGRAM ANALYSIS CHARTS

CHART B

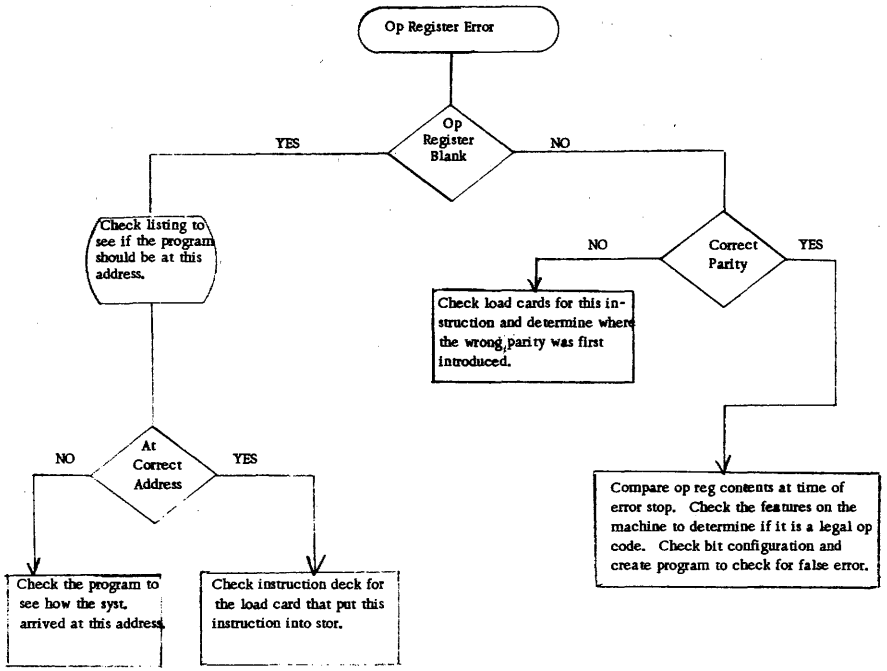


FIGURE C-3B
PROGRAM ANALYSIS CHARTS

CHART C

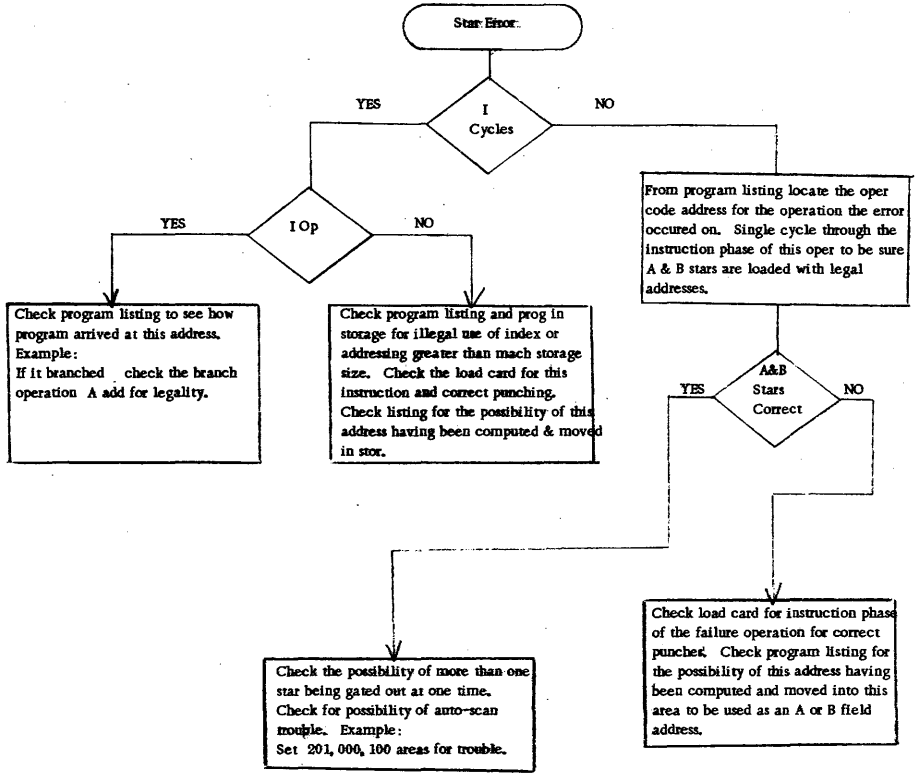
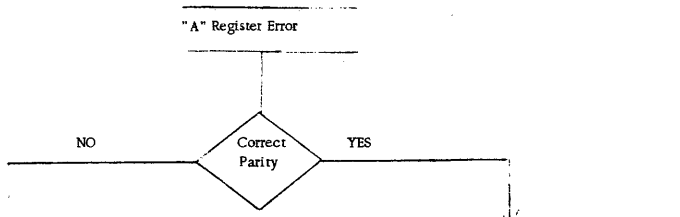


FIGURE C-3C
PROGRAM ANALYSIS CHARTS

CHART D

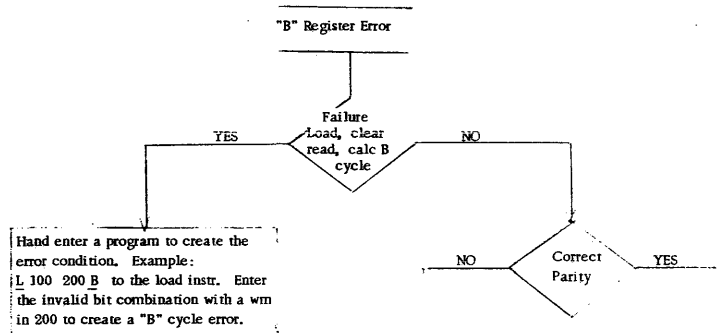


Usually the A register is gated in from the B register. In this case, there should also be a B reg error. The procedure for this is described under B reg error. If the character has been set into the A reg by arithmetic, read or punch controls loop the existing prog or create a program duplicating the condition or run the CE diagnostic deck for trouble shooting.

Loop system by either altering the program or hand entering a program to create the error condition (same bit structure). Example: L 100 200 B load and branch back to the load instruction. Asume the bit combination 42 wm causes an A register error. Address 100 should contain 42 wm. This would create the assumed error card.

FIGURE C-3D
PROGRAM ANALYSIS CHARTS

CHART E



Hand enter a program to create the error condition. Example: L 100 200 B to the load instr. Enter the invalid bit combination with a wm in 200 to create a "B" cycle error.

Loop system either by looping the program or by hand entering a prog to create the error conditions. Example: M 100 200 B assume that the bit combination 421 causes a "B" register error. Address 100 or 200 should contain 421 and address 099 or 199 could contain any legal bits with a wm. This would create the assumed error condition.

The B register is gated into from storage only, so determine whether the information was correct in storage. Find where this address was last entered into in the program. Single cycle through the instruction you find to see if the information was correct when you inhibited it. If it was, check the core array, sense lines & B reg latches. If it wasn't, loop this instruction or hand load a program to create this error condition.

FIGURE C-3E
Program Analysis Charts

(C-4) Process Unit Error Conditions

Figure C-4 lists the 1401/60 error stop conditions and the associated reset and/or re-start procedures.

Unit	Type of Error	Machine Stops Process Check Stops "ON"	Storage Adr. Reg. Contains	Lights ON When Stopped	Reset By	Remarks
A Reg	Parity	End of Next Cycle (B Cycle)	"B" Address	Process A Reg Check Reset	Check Reset Key	Contents of A Reg at time of error will still be on display (A Reg resets on A cyc. only)
B Reg	Parity	End of Cycle in Which Error is Detected	Address of loc. that was read into "B" Reg	Process B Reg Check Reset	Check Reset Key	Contents of B Reg at time error is detected will remain on display in B Reg.
Arith	Validity	End of Following Cycle	Normally 1 less than the loc. that resultant is in.	Process Logic Check Reset	Check Reset Key	Adr. Reg will indicate one less than the loc. that the resultant is read into except: 1) When error is detected in the last cyc. of the 1st forward scan on a recomplement operation when it will indicate the same loc. or 2) It will indicate one more than the loc. the result is read into on a reverse scan operation. The bit combination which caused the error will be in the storage unit and not on display under "logic." Remember it is quibinary form when checked and goes thru the translator before going into storage.
Inhibit Switching	Parity	End of Following Cycle	Dependent on operation being performed and phase that system is in.	Process Stor. Check Reset	Check Reset Key	This type of error indicates that an even bit configuration has been read into storage.

Figure C-4 a PROCESS UNIT ERROR CONDITIONS

Unit	Type of Error	Machine Stops Process Check Stops "ON"	Storage Adr. Reg. Contains	Lights ON When Stopped	Reset By	Remarks
Op Reg	Validity & Parity	End of Cycle in which error is detected	Dependent on type of operation being performed and phase.	Process Op Reg. Check Reset	Check Reset Key	The check latch will not turn on during 1 Ring Op time
Storage Address Register	Parity & Validity	End of Cycle in which error was detected.	Bit combination that caused error.	Process Storage Address Check Reset	Check Reset Key	The error check is made after the address is serialized. An error could be caused by a fault in serializing.
	Wrap Around	End of Following Cycle	Dependent on operation being performed & modification.	Process Storage Address Check Reset	Check Reset Key	Can be modified by +1 or - 1
						NOTE: If any of the above errors occur during an input/output operation, the system will complete the particular operation involved before stopping.

Figure C-4b PROCESS UNIT ERROR CONDITIONS

(D) Reader/Punch Operating Pointers and Miscellaneous Error Indications

(D-1) Start and Stop Keys

The Start and Stop keys on the IBM 1440 System are not common. Each key applies only to its respective unit. This differs from the 1401 where any stop key can stop the system.

A condition exists where the operator cannot stop the machine from any stop key. This happens when:

1. Using reader punch release
2. Last card testing
3. Branch on last card to other than reader punch operation

The problem occurs when a start read feed (SRF) or start punch feed (SPF) is given after the last card has been read. To prevent this from happening, test for last card prior to issuing a SRF or SPF command. Bypass the SRF or SPF on last-card-on condition.

(D-2) 1440 I/O Operation

1. When attempting to punch or print without a termination group-mark after the last core position accessed, the unit addressed will go out of its ready status. Manual depression of the start on that unit will be required before it can operate again.

2. The punch skip feature increases available process time, but does not make the actual skip go faster than normal speed.

3. Since cards follow the same path for both reading and punching, the possibility of lacing a deck of cards is greater than normal. All decks should be reproduced to have a back-up, especially while testing.

4. A punch area of more than 80 positions will cause column 81 to be laced.

5. If no GMWM is present within 81 positions of the card read area, the reader may continue to read cards and data will be entered serially into core until either a GMWM is encountered or until core has been wrapped.

(D-3) Stacker Selection (1402)

Normally, the d-character for a stacker select op would be 1, 2, 4 or 8. However, the instruction SS 5 will cause both read and punch stacker selection. This instruction must be given within 10 MS after a read op.

The instruction SS 5 causes both a read stacker latch and a punch stacker latch to be set. Once such a latch is set, it can only be released by completing the respective stacker selection.

Consequently, the next card from the punch feed will go to the number 4 pocket, even though a number of cards have been read following the selection of the reader card into pocket 1.

The various stop keys are inoperative during a stacker selection operation, so that once the card is mechanically selected (in motion), selection will be completed before the machine stops.

(D-4) Punch Stacker Selection with I/O Check Stop Switch Off

When correct cards are being selected to either the 4 or 8 pocket and a punch error occurs, it is possible that the first good card after the error card will be selected in the normal punch pocket. Program steps to avoid this false selection are: (See Figure D-1)

1. Punch op (or PFR op).
2. Branch on Punch Error, set switch 1 on, branch to SELECT.
3. SELECT (stacker select op)
4. Test switch 1: if on, perform correct steps, if off, NSI.
5. NSI (Next sequential instruction).

This allows correct (Normal Punch Pocket) selection of errors but allows program-controlled selection of all good cards.

Note: In many cases, it will be possible to perform the corrective steps in the time allowed, without programming a switch 1.

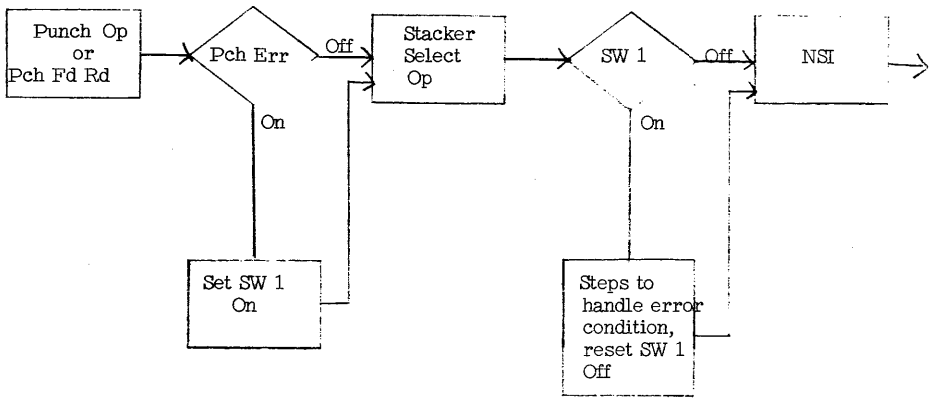


Figure D-1
Stacker Selection with I/O Off

(D-5) 1440 Stacker Select and Branch Instruction

The Stacker Select and Branch command is not provided for the 1440.

(D-6) Last Card Indication (1442)

The 1440 last card indicator is turned on when the following conditions are met:

1. The feed hopper is empty.
2. A card is ready to be read.
3. The start key on the 1442 has been pressed.

When punching into cards that have been read on the same pass, the next-to-the-last card has been read when Ready Status is dropped. As the punch instruction is executed, the operator must press the 1442 start key, which sets the Last Card Switch on. If the last card test precedes the read, the last card will not be read.

As long as there are cards in the Read-Punch feed path, a read instruction can be executed. If the last card is at the punch station, a read command may be used to run the last card into the stacker. When a read is executed under this condition, blanks are entered into the read area: MR %G1, 901

(D-7) 1402 Punch-Feed Notes

1. Sense switch A does not test for last card in the punch feed, even when performing a Punch Feed Read op. A test for punch feed last card must be programmed, such as testing for a specific character in a particular column.

2. Programs cannot be loaded from the punch side, even when the Punch Feed Read special feature is installed.

3. PFR and normal read operations use some of the same read circuits of the 1402. Therefore, a PFR error can be tested by using the read error branch and appropriate d-modifiers, when the branch follows the punch feed read op. Punch errors can similarly be recognized.

4. At (punch) end of job, clear the punch area (101-180) and program another dummy punch cycle. This will permit proper stacker selection of the last validly punched card, and insure that all good cards are out of the punch feed. The stations of the punch feed will contain blank cards, ready for the next job. It is permissable to have blank cards in the punch feed at the start of any punch feed run , except a PFR (Punch Feed Read). However, it is normally good practice to press both reader and punch non-process run-out keys to insure that the reader and punch clutches are latched. This procedure will prevent reader and punch stop errors when starting on the next program.

If it is essential that hole patches be used to correct error punching in original documents, place them on the back side of the card. The 1402 feed cards face down. This allows the 1402 punch to repunch the hole(s) from the non-sticky side of the patch. If this practice is not followed, the sticky side of the patches will build up on the business end of the punch, and cause excessive wear.

Apply patches to error punches only, and not to the entire field. Do not apply more than one patch to any one punch hole: don't overlap patches. This is a common cause of punch feed jams.

(D-8) 1402 Error Conditions

Figure D-2 lists the 1402 error stop conditions and the associated restart procedure.

Unit	Error	Machine Stops-I/O Check Switch "ON"	Lights ON When Stopped	Reset By	Remarks
Reader	Read Check	At the end of the feed cycle.	Read Check (1402) Read (Process)	Check Reset Key on the 1402.	Cards must be run out before check reset key becomes effective.
Reader	Validity	At the end of the feed cycle.	Validity (1402) Read (Process)	Check Reset Key on the 1402.	Cards must be run out before the check reset key is effective.
		NOTE: Also, if the invalid combination causes incorrect parity:			
			Storage (Process) Process (process) Check Reset Process	Check Reset on the Process Unit.	
Reader	Jam	At the end of the feed cycle.	Reader Stop (1402) Read (Process Unit)	Check Reset Key (1402)	Cards must run out before reset key is effective. Damaged cards must be repaired.
Punch	Punch Check	At the end of the feed cycle.	Pch. Check (1402) Punch (Process)	Check Reset Key (1402)	
Punch	Parity	At the end of the feed cycle.	Punch (process) Process (") B Reg (") Check Reset (Process)	Check Reset Key	
Punch	Jam	At the end of the feed cycle.	Punch Stop (1402) Punch (Process)	Check Reset Key (1402)	Cards must be run out before the reset key is effective.

Figure D-2 1402 Error Conditions

(D-9) 1402 Punch Feed Restart Procedure

In any application which includes a requirement for summary totals to be accumulated from fields being punched into each card, restart procedures are more involved than normal.

Basically, the increased complexity is caused by the physical arrangement of the card stations in the punch side of the 1402. That is, by the time a punch error has been detected, the card behind the one in error has also been punched. Consequently, any restart procedure must allow for the "backing out" of the amount fields from the accumulators for both cards involved. The amount of programming necessary to accomplish this will vary with the individual program.

The use of the Punch Feed Read Special Features on such applications complicates the necessary restart procedures even more; therefore, 1401's which include this feature should be given special attention in this respect.

Adequate restart procedures must be included in the original writing of such programs in order to minimize difficulties.

(D-10) Punch Feed Read Feature Re-Start Procedures

1. PFR Validity Error: (Read side validity error light on)
   The first card into the stacker after run-out will have been punched but not checked.

   The second card (B) is invalid. It has been read but not punched or checked.

   The third card (C) has been neither read, punched nor checked.

   Restart:

   1. Remove cards from punch hopper and stackers. Run out remaining 3 cards into the stacker.
   2. Determine the error in card B.
   3. Repunch the original data into a new card A. (Do not include any punches produced by the 1402 during this pass).
   4. Correct and re-keypunch the original data in card B.

   5. Reset the error with the 1402 check reset key (and process unit check reset key if the punch read error caused a process light).
   6. Restart with cards A, B, and C, followed by cards previously removed from the punch hopper.

2. PFR Punch Check Error (hole count error): The last card stacked (A) is the card in error.

   The first card (B) into the stacker after run-out has been punched but not checked.

   The second card stacked (C) has been read, but not punched or checked.

   The third card stacked after run-out (D) has not been read, punched or checked.

   Restart:

   1. Remove the remaining cards from the punch hopper. Run out the 3 cards left in the machine into the stacker.
   2. Determine the error in card A (last card stacked before the machine stopped with the punch check light on).
   3. Re-keypunch the original data into a new card A. (Delete any punches produced by the 1402 during this pass).
   4. Re-keypunch the original data from card B into a new card B deleting any data punched by the 1402 during this pass.
   5. Reset the punch check light with the 1402 check reset key. There should be no process error, unless there was a compound error (punch check and validity errors).
   6. Restart with cards A, B, C, and D, followed by cards previously removed from the punch hopper.

These procedures assume that the job did not have cards feeding from the read feed. If there were, these cards must be run out with the non-process runout key, and re-fed. In some cases, it will be necessary to back the job up (or restart), depending on the card order and the application.

3. Alternate Procedure for Clearing Punch Checks on 1401 with Punch Feed Read Feature

On updating programs, both tape and disk, when data cards are being processed and punched in the punch feed, a punch check can disturb the updating sequence of the run. The punch check is detected after the card in error has updated the file, and the following card has been punched and (depending on the program logic) may have also updated the file. Any attempt to reconstruct would be time consuming since it would have to be done before processing could continue.

A simple procedure for restarting appears to preserve the updating sequence in all cases. The procedure is as follows:

1. Remove the cards from the punch hopper.
2. Press the non-process runout key to clear three cards from the punch.
3. Of the last four cards in the punch stacker, the first is the one which caused the punch check, the second has been punched but not checked, the third has been read but not processed, and the fourth has not been read.
4. To restart, take a blank card (preferably of a different color or corner cut from the data cards) and place this in the punch hopper followed by the third and fourth (the last two) cards from the punch stacker.
5. Replace the remainder of the input cards in the punch feed.
6. Press the check reset and start button on the 1402. Do not press start reset on the 1401 console.
7. After the run, the odd color card can be discarded and the two cards in front of that card should be checked manually.

(D-11) Punch Check Error, Using Pre-Punched Cards

When using pre-punched cards in the 1402 without the Punch Feed Read feature, a punch hole count error is developed unless the pre-punched holes are repunched.

On the 1402 with the Punch Feed Read feature, pre-punched holes must not be repunched.

In either case, additional holes may be punched by the 1402 in the pre-punched columns. Many punch hole combinations cannot be punched without the Column Binary feature. Thus: without PFR, if a card column already has a 7 punch, and the character R(BCD code B, 1, 8) is required in this column in addition to the existing 7 punch (BCD code 4, 2, 1), a punch check condition will still result. The CPU cannot store a 7 (4, 2, 1) and an R(B, 8, 1) in the same storage location.

Consider that a 5 has been pre-punched in column 15. Without the PFR feature, an X zone (BCD code of B-bit only) may be punched in column 15, but the 5 must be repunched. This would be done by storing an N(BCD code B, 4, 1) in the storage position that will be transferred to card column 15. With the PFR feature, an X may be punched in column 15, but the 5 must not be repunched.

(D- 12) Validity Errors

If a branch on reader error instruction is being used, a branch on process error instruction should also be used in the same routine when a 1407 console inquiry station is on-line. (The I/O check stop switch on the 1401 and the process check stop switch on the 1407 must be off to make these instructions effective.)

A validity error can cause a process error. Since the process check stop switch is off, there should be a test for process error following the reader error test. A test for process error should also precede the read op. This will eliminate confusion about the cause of the error stop.

Any time the I/O check stop switch is off, tests for all I/O error conditions must be made. Thus: if card errors are expected, reader check and validity errors will be caught, as well as any other I/O errors (printer sync or print check).

(D- 13) Collating into the 8/2 Pocket

There are various applications in which the collating of cards from the read and punch feeds into the 8/2 pocket of the 1402 will save a subsequent off-line collating operation.

Because of a difference in card speeds between the two feeds and the fact that the punch feed has an additional card station, the program must be geared to insure proper card control.

To effect this collating operation, the program must include the following two provisions:

1. A "dummy" punch cycle must be taken to pass the card through the check brushes and start it on its way to the 8/2 pocket. This is facilitated by placing a blank card behind each card in the punch feed that is to be directed to the 8/2 pocket.

2. Following the punch-complete of the "dummy" punch cycle, 143 ms should elapse prior to the next read instruction to insure that the card from the punch feed reaches the 8/2 pocket.

   The 143 ms figure has been arrived at by the following analysis. From the time of the punch complete on the punch cycle that the card passes the check brushes until the card is at the 8/2 pocket, 360 ms elapses. (Punch complete is that point in the punch cycle that the 1401 interlock is released and the next program instruction can take place.) It takes 217 ms from the time the read instruction is given for the card to be read and then travel to the 8/2 pocket. The difference is 143 ms.

   To determine the 143 ms, calculate the basic-loop time (shortest path) required to execute the actual program instructions between the dummy punch cycle and the next card read instruction. This minimum constant is determined from the instruction timing information in the System Operation Reference Manual or in the System Instruction and Timing Summary for the system to which the 1402 is attached.

   If the basic-loop time constant is 143 ms or less, a delay-loop subroutine (Figure D-3) is required to avoid false merging and jamming. (Allow somewhat more than 143 ms before reading the next card, as a safeguard against timing variances.) (Figure (D-4)

LABEL	LENGTH	OP	OPERAND	
	07	ZA	STCONS, CTR	Clear counter and add first time.
STALL	07	A	STCONS, CTR	Add constant to CTR.
	05	B	(MAINRT), Z	Branch to main routine on overflow.
	04	B	STALL	Branch back to stall loop.
CTR	02 or 03	DCW	*	Allow 2 or 3 positions according to table.
STCONS	01	DCW	*	Insert digit 1 through 9 according to table.

Figure D-3
Delay Loop Subroutine

Stall Constant	Using 3 Position "CTR"		Using 2 Position "CTR"	
"STCONS"	No. of Adds	Stall Time in MS	No. of Adds	Stall Time in MS
1	1000	287.293	100	27.393
2	500	143.543	50	13.593
3	334	95.818	34	9.177
4	250	71.668	25	6.696
5	200	57.293	20	5.313
6	167	47.805	17	4.485
7	143	40.905	15	3.933
8	125	35.730	13	3.381
9	112	31.993	12	3.105

Storage Requirements	3 Position "CTR"	2 Position "CTR"
SUBROUTINE	23	23
STALL CONSTANT (STCONS)	1	1
COUNTER (CTR)	3	2
	27 positions	26 positions

Figure D-4
"STALL" Timing Table

(E) Printer Operation Pointers and Miscellaneous Error Indications

(E-1) 1403 Error Conditions

Figure E-1 lists the 1403 error stop conditions and their associated re-start procedures.

Error	1401 Stops (I/O Check Stop Switch On)	Lights On		Reset by	Remarks
		Process Unit	Printer		
Parity	Upon completion of print out.	B. Reg process		Check Reset (Process Out)	
Type Sync	Upon completion of print out.	Printer	Sync Check	Check Reset (Printer)	
Hammer Fire	Upon completion of print out	Printer	Print Check	Same	Sets Error Store Core
Print Line Complete	Upon completion of print out	Printer	Print Check	Same	Sets Error Store Core

Figure E-1, 1403 Error Conditions

(E-2) Printer errors

There are four printer errors on the 1403:

1. Sync check (Sync Check Light), 1403 chain out of syncronization with the 1401 timers.

2. Storage address check (Print Check Light), correct storage location has not been addressed for printing.

3. Print Line Complete Check (Print Check Light), each core position in the print area has not been scanned.

4. Hammer Check (Print Check Light), hammer firing either was not called for and occurred, or was called for and did not occur.

These checks insure that the following conditions have been met:

a. The correct character has been printed in the correct print position.

b. All printable characters have been printed.

c. Printing did not occur for unprintable characters.

d. A hammer did not fire more than once for one print position, for any one print line.

A valid CPU bit configuration that is an unprintable character on the 1403 will not cause a printer error.

However, if a printable invalid character is printed, an error will occur. (If the character bit configuration was B, A, 4, 2, and 1 but the A-bit was missing: the character Q would have printed for the alphabetic G intended.)

An automatic single space will follow a line of error printing, unless this line was printed using the space suppress feature.

(E-3) I/O Error Checking

The point reached in an I/O Operation can be determined if there is a sync check error. First, consider a print only operation. If a sync error occurs during printing, the printing will be completed before the 1401 stops. If the sync check occurs between print cycles, the 1401 will stop before the next line is printed.

In the first case, without print storage, the B-STAR will contain 335 (or 303 for a 100 char print span) when the 1401 stops. In the second case, B-STAR will contain 201.

In combination I/O Op's, the read and/or punch portion of the Op will not be executed if the sync check occurs during printing, but before the read or punch start begins. If the error occurs after the reader or punch has started, these portions of the Op will be completed. If Print-Read is executed, the B-STAR will contain 081 when the 1401 stops. If Print-Punch, or Read-Print-Punch is executed, the B-STAR will stand at 181 when the 1401 stops.

If the Sync Check and Print Check are both ON, the Sync Check is not considered a print error. If the chain is out of time with the printer circuits, the machine stops before the next print Op. In this case, the sync check light will be on, but the print check light will not be on.

(E-4) Printer Notes (1403)

1. If the branch on printer error instruction -- BIN III, (I/O check stop switch off) is given immediately following a print instruction (print storage feature), the system will interlock until printing is complete (not including spacing).

   To prevent this loss of process time, use the branch on printer busy instruction -- BPB III. Program processing can continue until the printer is no longer busy. Then the error latch can be interrogated.

2. With print storage, put in the following loop before testing for carriage overflow -- BPB *-4.

3. Whenever possible, use a delayed rather than immediate spacing operation. This technique saves machine time.

(E-5) 1440 Console Printer

1. A left bracket (BA841) will cause carriage tabulation and a right bracket (CB841) will cause a carriage return operation if these characters appear within the data to be printed from core storage. The character causing the tab or carriage return will not be printed in the output. This can be convenient when using the console printer in some sort of formated output printing.

2. To space up the typewriter carriage, execute a write console typewriter instruction with the operand the address of a GMWM.

3. If the inquiry indicator latch has not been set on before a read from the console typewriter instruction, the instruction will act as a NO-OP.

(E-6) 1443 Printer Pointers

1. 1443 Printer Carriage Function - the carriage cannot be manually restored or spaced unless the 1443 stop key has been pressed to take the printer out of ready status.

2. 1443 Printer Hang-Up - if the system interlocks because the printer is not in ready status, the printer light on the console does not come on. The only indication is an M in the Op Register and a W in the A-register.

3. 1443 Paper Drag Level - if the printer frequently drops ready status, check the Paper Drag Indicator. It should be set between 1 and 2.

4. In storage print out mode, 144 characters (on model 2) beginning with the address in the manual address switches will be printed on the 1443. If:

   0001 is dialed, 0001 to 0144 prints
   0002 is dialed, 0002 to 0145 prints
   etc.

(E-7) 1440 Carriage Control and Branch Instruction

The Carriage Control and Branch command is not provided for the 1440.

(E-8) Forms Skipping

1. Continuous skipping will result if a skip-to instruction is given while the 1403 is in the process of executing a skip to that same carriage tape channel.

   This condition might arise where the programmer has called for a skip to some specific channel on more than one condition, but has neglected to allow time for this skip to be completed before another skip is initiated from another section of the routine.

2. If a skip delay precedes a skip immediate (to another carriage tape channel) in the same processing interval, the delayed skip is cancelled.

   If a skip immediate precedes a skip delay command (to another channel), both skips will be executed in the normal manner.

3. If an invalid forms Op d-modifier is used, the carriage will skip continuously. (In this case, skipping can be stopped by pressing either the CPU start reset key, or the printer stop key.)

(E-9) 1403 Forms Specifications

The degree of acceptability for the particular job will dictate the grade of paper and carbon used. The printing requirements between the IBM 402 and 407 vary somewhat, as between either of these machines and the chain printer.

1. An original and 3 acceptable copies can be obtained, using 11-13 lb. continuous #4 Sulphite bond paper and 7-9 lb. Kraft with soft, medium and hard carbon, as produced by various forms companies.

2. An original and 5 acceptable copies can be obtained, using 11-13 lb. continuous #4 Sulphite bond and 7-9 lb. Kraft carbon selected to provide the desired printing.

3. The use of premium paper will make the printing of six copies easier. The relation between the paper and the type of carbon for the last copy is of special importance to reduce the hammer face impression obtained on all back-printing machines. (It is interesting to note that some paper manufacturers feel that soft carbon is the best, while others recommend hard carbon.) Many other grades and weights of paper can be used, depending on the application and the desired print quality. Excellent eight-copy printing has been obtained using a very low cost premium paper from one manufacturer. A 12-part printed form with carbonized backing instead of individual carbon sheets has given good results. Thick packs or stiff forms are difficult to print on the chain printer, and should be avoided. Samples of a three-part form, two of which were card stock, have been printed with good results.

4. The thickness and stiffness of the pack is the limiting factor when used with any on-the-fly type printer. Minimum weight paper for single sheet work is not recommended, as light weight papers are subject to adverse static conditions.

5. Multilith masters can be cut, either with or without a ribbon. It should be tried both ways on the particular type of master paper being used.

6. Loose staples can cause jamming. However, generally good results have been experienced, especially when the staples are placed horizontally. Care must be taken to insure that printing does not occur on the stapled area, as the type face will be damaged.

(F) Branch Instruction Pointers

(F-1) Branch if Indicator On Instructions

1. If a Branch if Indicator On instruction contains a d-modifier not used by the machine, the resulting instruction is effectively a NOP. The next sequential instruction is performed.

2. With the Advanced Program Feature installed, an automatic function of a successful branch is to save the address of the NSI. This address is placed in the B-STAR. This operation adds one storage cycle to the time for all branch instructions. However, only successful branch instructions will actually use this additional time. The address which has been placed in the B-STAR can be stored in any other valid CPU address by using the SBR (H) op code as the first instruction of the branched-to address. The location to which NSI address is usually stored is the I-Address of the subroutine exit-branch instruction.

3. If the advanced programming feature is not present, the contents of the B-STAR (NSI address) are erased when a successful branch occurs. This does not increase the process time for this op. This erasure occurs only if the branch is successful.

(F-2) Branch On Character Equal Instructions

1. A 7-position branch instruction can act like an 8-position Branch On Character Equal command. The character in the units position of the B-address is retained in the A-register, and therefore acts as the d-modifier for the branch.

   This can be used to advantage. If a branch is required when core storage position 079 contains a 3, use the instruction: B III 079. If position 079 does not have the character (9), the NSI will be executed. Obviously, this technique has limited use and applies only when the units position of the location corresponds to the code to be examined.

2. When a code within a record is to be tested against a series of acceptable possibilities, use the following method:

Label	Op	Operands
TEST	MCW	(code), *+8
	BCE	XXX, ZZZ, ?
	BCE	
	BCE	
	BCE	
	NSI	

   (18 positions used + 4 position constant)

   ? designates the code moved into the first BCE instruction and assumes a W/M. ZZZ is the location of a constant that contains a list of possible acceptable codes, such as: DCBA. XXX is the branch-to I-address.

   This procedure represents a substantial savings of core storage if the possibilities table is lengthy, and where each BCE command tests the same code positions of the record against another character of the table. The normal program would be:

```
Label       Op          Operand
TEST        BCE         XXX,  YYY,  A
            BCE         XXX,  YYY,  B
            BCE         XXX,  YYY,  C
            BCE         XXX,  YYY,  D
            NSI
```

(32 positions used)

Where YYY is the location of the code to be tested.

If a W/M is not present at "code", use the following routine:

```
Label       Op          Operand
            MN          (CODE),  TEST + 7
            MZ          (CODE),  TEST + 7
TEST        BCE         XXX,     ZZZ,  ?
            BCE
            BCE
            BCE
            NSI
```

(25 positions + 4 position constant)

(F-3) Branch On Access Busy Instruction

The Branch Access Busy (B III/) is not effective if given following a Seek command. It must follow either a Read Disk or Write Disk instruction. This branch command has the effect of a branch if an access-busy condition has prevented the data transfer.

One exception to the above is when a seek is followed directly by another Seek. Here, the Branch Access Busy command must follow the second Seek command and branch to that seek. On machines without Seek-Overlap, this is necessary to prevent bypassing the second Seek.

(F-4) Saving Branch Instructions

When a series of tests are to be made and a different resultant action taken with a return to a common point, the execution of the action prior to the test will save branch instructions:

Normal - Test Then Action

```
Label       Op      Operand
            BCE     DIGIT, CODE, 2
            BCE     ZONE, CODE, 6
            -
            B       ERROR
            -
DIGIT       MCN     CHARAC, WORK
            B       GO
ZONE        MCW     RECORD, WORK
            B       GO
```

Save Branches - Action, Then Test

```
Op      Operand
MCW     CHARAC, WORK
BCE     GO, CODE, 2
MCW     RECORD, WORK
BCE     GO, CODE, 6
-
B       ERROR
```

(F-5) Testing Sense Switch Settings by Pivotal Branch Technique

With the availability of 6 sense switches, each of which can be on or off, it is possible to set $2^6 = 64$ possible combinations of sense switches. To test these possibilities, the following routine should be useful:

Label	Op	Operand	Comments
	SBR	X1, 63	Place constant "63" in index reg. 1
	BSS	GON, G	Is SSW G on?
	SBR	X1,99+X1	Add 100's complement of 1 to ind. reg. 1
GON	BSS	FON, F	Is SSW F on?
	SBR	X1, 98+X1	Add 100's complement of 2 to ind. reg. 1
FON	BSS	EON, E	Is SSW E on?
	SBR	X1, 96+X1	Add 100's complement of 4 to ind. reg. 1
EON	BSS	DON, D	Is SSW D on?
	SBR	X1, 92+X1	Add 100's complement of 8 to ind. reg. 1
DON	BSS	CON, C	Is SSW C on?
	SBR	X1, 84+X1	Add 100's complement of 16 to ind. reg. 1
CON	BSS	BON, B	Is SSW B on?
	SBR	X1, 68+X1	Add 100's complement of 32 to ind. reg. 1
BON	MCW	@0@, X1-2	Move a zero to high order of ind. reg. 1

Index Register 1 now contains a number which describes the sense switch settings. This number is now used to modify the pivotal branch to one of 64 possible routines:

Label	Op	Operand	Comments
	A	X1	Adds Ind. Reg. 1 to itself (2X IX1)
	A	X1	Adds Ind. Reg. 1 to itself (4X IX1)
	B	SSWTBL+X1	This is the pivotal branch
SSWTBL	B	ALL OFF	All sense switches off
	B	G	Sense Switch G on, B, C, D, E, F Off
	B	F	Sense Switch F on, B, C, D, E, G Off
	B	FG	Sense Switch F, G on, B, C, D, E Off
	B	E	Sense Switch E on, B, C, D, F, G Off
	B	EG	Sense Switch E, G on, B, C, D, F Off
	B	EF	Sense Switch E, F on, B, C, D, G Off
	B	EFG	Sense Switch E, F, G on, B, C, D Off
	B	D	Sense Switch D on, B, C, E, F, G Off
	B	DG	Sense Switch D, G on, B, C, E, F Off
	B	DF	Sense Switch D, F on, B, C, E, G Off
	B	DFG	Sense Switch D, F, G on, B, C, E Off

Label	Op	Operand	Comments
	B	DE	Sense Switch D, E on, B, C, F, G off
	B	DEG	Sense Switch D, E, G on, B, C, F off
	B	DEF	Sense Switch D, E, F, on, B, C, G off
	B	DEFG	Sense Switch D, E, F, G, on, B, C off
	B	C	Sense Switch C on, B, D, E, F, G off
	B	CG	Sense Switch C, G on, B, D, E, F off
	B	CF	Sense Switch C, F on, B, D, E, G off
	B	CFG	Sense Switch C, F, G on, B, D, E off
	B	CE	Sense Switch C, E on, B, D, F, G off
	B	CEG	Sense Switch C, E, G on, B, D, F off
	B	CEF	Sense Switch C, E, F on, B, D, G off
	B	CEFG	Sense Switch C, E, F, G on, B, D off
	B	CD	Sense Switch C, D on, B, E, F, G off
	B	CDG	Sense Switch C, D, G on, B, E, F off

Label	Op	Operand	Comments
	B	CDF	Sense Switch C, D, F on, B, E, G off
	B	CDFG	Sense Switch C, D, F, G on, B, E off
	B	CDE	Sense Switch C, D, E on, B, F, G off
	B	CDEG	Sense Switch C, D, E, G on, B, F off
	B	CDEF	Sense Switch C, D, E, F on, B, G off
	B	CDEFG	Sense Switch C, D, E, F, G on, B off
	B	B	Sense Switch B on, C, D, E, F, G off
	B	BG	Sense Switch B, G on, C, D, E, F off
	B	BF	Sense Switch B, F, on, C, D, E, G off
	B	BFG	Sense Switch B, F, G on, C, D, E off
	B	BE	Sense Switch B, E on, C, D, F, G off
	B	BEG	Sense Switch B, E, G on, C, D, F off
	B	BEF	Sense Switch B, E, F on, C, D, G off
	B	BEFG	Sense Switch B, E, F, G on, C, D off
	B	BD	Sense Switch B, D on, C, E, F, G off

Label	Op	Operand	Comments
	B	BDG	Sense Switch B, D, G on, C, E, F off
	B	BDF	Sense Switch B, D, F on, C, E, G off
	B	BDFG	Sense Switch B, D, F, G on, C, E off
	B	BDE	Sense Switch B, D, E on, C, F, G off
	B	BDEG	Sense Switch B, D, E, G on, C, F off
	B	BDEF	Sense Switch B, D, E, F on, C, G off
	B	BDEFG	Sense Switch B, D, E, F, G on, C off
	B	BC	Sense Switch B, C on, D, E, F, G off
	B	BCG	Sense Switch B, C, G, on, D, E, F off
	B	BCF	Sense Switch B, C, F on, D, E, G off
	B	BCFG	Sense Switch B, C, F, G on, D, E off
	B	BCE	Sense Switch B, C, E on, F, D, G off
	B	BCEG	Sense Switch B, C, E, G on, D, F off
	B	BCEF	Sense Switch B, C, E, F on, D, G off
	B	BCEFG	Sense Switch B, C, E, F, G on, D off

Label	Op	Operand	Comments
	B	BCD	Sense Switch B, D, C on, E, F, G off
	B	BCDG	Sense Switch B, C, D, G on, E, F off
	B	BCDF	Sense Switch B, C, D, F on, E, G off
	B	BCDFG	Sense Switch B, C, D, G, F on, E off
	B	BCDE	Sense Switch B, C, D, E on, F, G off
	B	BCDEG	Sense Switch B, C, D, E, G on, F off
	B	BCDEF	Sense Switch B, C, D, E, F on, G off
	B	BCDEFG	All sense switches on

The program must be written in this sequence to work properly.

(G) Add and Subtract Instruction Pointers

(G-1) Arith Overflow Indicator

This indicator is turned on by most object-deck clear-storage routines. If this indicator is to be tested during the program, it must first be reset off by a dummy test.

A method to reset this indicator is to place a dummy branch immediately ahead of the instruction in the program which may cause an overflow, or as a housekeeping instruction: BAV * + 1.

Note: The accumulator field must be at least two positions for the overflow indicator to be effective. It is not set on when a single-position field has an arithmetic overflow.

(G-2) Field Reset, Using the Subtract Op

Any field defined by a high order word mark can be reset to zeros with the single-address subtract op instruction --S (AAA).

Notice, however, that the units positions of the field will be signed. If the original field was negative, the resulting sign will be minus; if the original sign was positive, the resulting sign will be plus. In many arithmetic functions, this is of little importance because of arithmetic sign control.

An exception to this is the reseting of index registers. These fields must be left unsigned. However, an index register can be reset using the single-address subtract op. This technique is illustrated in the subroutine section of this series.

The field sign must be considered when the field is compared. It will compare unequal to a field containing all zeros, for instance.

If the sign of the field is constant (always plus or always minus), this zone can be used as a program constant any time such zone-only information is required.

(G-3) Miscellaneous Addition Notes

1. Accumulators

   Where possible, define core storage counter areas for each total level as adjacent fields. This facilitates chaining and/or indexing operations affecting these fields.

2. Adding a constant 1 without a constant

   To add 1 to a counter without previously defining the constant 1:

Op	Operand
A	*-6, CTR

   *-6 refers to the op code "A" which is a + 1.

3. Zero and Add Instruction

   The A-field of the Zero and Add instruction does not go through the adder. Blanks in the original A-field will remain blanks in the resultant B-field. In any addition or subtraction subsequent to executing the ZA Op, these blanks will be treated as zeros.

   If the A-field is shorter than the B-field, zeros will be inserted to fill out the B-field.

   The zero and subtract instruction parallels the ZA function except for sign control.

4. Resetting a counter to +0 without a constant

   To clear a counter to +0 with a +0 constant:

```
ZA  *-6, CTR
```

   *-6 refers to the op code "ZA", which is a +0.

(G-4) B-Field Sign After an Add or Subtract Operation

Figure G-1 shows the resultant sign of the B-field after add or subtract operations.

	Unsigned B Field			+ Sign B Field			- Sign B Field		
	If Value of			If Value of			If Value of		
	A < B	A = B	A > B	A < B	A = B	A > B	A < B	A = B	A > B
ADD UNSIGNED A FIELD	NS	NS	NS	+	+0	+	-	-0	+
ADD + A FIELD	NS	NS	NS	+	+0	+	-	-0	+
ADD - A FIELD	+	+0	-	+	+0	-	-	-0	-
SUBTRACT UNSIGNED A FIELD	+	+0	-	+	+0	-	-	-0	-
SUBTRACT + A FIELD	+	+0	-	+	+0	-	-	-0	-
SUBTRACT - A FIELD	NS	NS	NS	+	+0	+	-	-0	+

Figure G-1. B-Field Sign after an Add or Subtract Operation

(G-5) Sign of Single Position Counters

When setting up a single position counter for iteration count control, or any other use, the following sign changes occur:

B-FIELD CHARACTER ORIG. SIGN	OPERATION	B-FIELD FINAL SIGN
Plus (BA)	Add	Plus (BA)
Minus (B)	Add	Minus (B) until counter is decreased past -0, then sign will remain plus. (BA)
No sign	Add	No sign
A-bit only	Add	A-bit only
Plus (BA)	Subtract	Plus (BA) until counter is decreased past +0, then sign remains minus. (B)
Minus (B)	Subtract	Minus (B)
A-bit only	Subtract	Plus (BA) until counter reaches plus zero, then sign remains minus. (B)
No sign	Subtract	Plus (BA) until counter reaches plus zero, then sign remains minus. (B)

These considerations are important when the result is to be compared, or is to be tested for a character-equal condition.

(G-6) Summary of Negative Zero Conditions

1. Reset subtract a + 0
2. Multiplication of two factors of opposite signs when one of these is zero.
3. Move zone of a "B" bit to the units position of a zero field.
4. Subtract with A field only (S AAA). Sign of the least significant digit remains the same. Thus, if the original field is minus, a minus zero will result.
5. Reducing a minus figure by repetitive additions of one until the figure reaches zero. This will be minus zero. Example: controlling an iterative routine.

(H) Multiply and Divide Instruction Pointers

(H-1) Multiple Multiplications at One Time

At least twenty five positions of core storage are used as preliminary steps to entering the multiply sub-routine described in the 1401 manual. To conserve core, (as well as time) a technique of calculating federal tax, fica tax, and city tax in one multiplication was devised. The various tax rates are placed in a work area designated as the multiplicand, (14000003625000000XX) where 14 is the federal tax rate, (14%) 3625 is the fica tax rate, (3 5/8%) and XX represents the city tax rate.* Gross pay is designated as the multiplier and since it is a five digit figure, five zeros are placed between each tax to prevent overlapping of results. For example:

	140000036250000015
X Gross	200.00
Equals	028.000007.250000003.00000
	federal tax ** fica tax city tax

From this point, it is simply a matter of determining the units position of each answer, and working on the results.

The same idea was used to calculate regular pay and premium pay in one multiplication by using regular hours and premium hours as the multiplicand, and rate as the multiplier.

* The left hand zero in all tax rates is assumed in decimal placement.

** In this method, the federal tax exemption for a weekly payroll equals: (13.00X number of dependents) X (.14), and is subtracted from the above answer to arrive at federal tax.

(H-2) Multiplication by Repetitive Addition

A more efficient use of CPU storage and processing time is often possible by using repetitive addition instead of the multiply special feature or a multiply-subroutine.

Many commercial applications involve multiplication of variable amounts by fixed constants. These constants might be a set percent as in payroll FIT and FICA; the factor 60 to convert hours to minutes; 12 to convert feet to inches; etc.

As an example, (Figure H-1) payroll federal withholding tax (FIT) of 14% is calculated, half-adjusted, and stored, using first the multiply feature, then the repetitive addition method.

FIELD NAME	MULTIPLY SPEC FEATURE			REPETITIVE ADDITION		
	Storage Size	Field Label	Actual Data	Storage Size	Field Label	Actual Data
Taxable Gross	5	TGROSS	12534	5	TGROSS	12534
Constant Five	1	FIVE	5	1	FIVE	5
FIT Percentage	2	PERCEN	14	–	–	–
Work Area	8	ACCUM	00000000	–	–	–
Output Area	5	STORE	01755	5 (2 Temporary positions needed)	STORE	01755
Total Positions	21			11		

Figure H - 1 - Multiply Example

A. Calculation with Multiply Feature

Position	Time	Op	A-Addr	B-Addr
7	.1380	ZA	PERCEN	ACCUM-6
7	.9660	M	TGROSS	ACCUM
7	.2070	A	FIVE	ACCUM-1
7	.2070	MCW	ACCUM-2	STORE
21(data positions)				
49 pos.	1.5180 MS			

B. Calculation using Repetitive Addition

Positions	Time	Op	A-Addr	B-Addr	Remarks
7	.2300	ZA	TGROSS	STORE+2	TGROSS X 1
4	.2415	A	STORE + 2		TGROSS X 2
4	.2415	A	STORE + 2		TGROSS X 4
7	.2415	A	TGROSS	STORE + 1	TGROSS X 14
7	.1955	A	FIVE	STORE + 1 (See Note)	
11(data positions)					
40 pos.	1.1500 MS				

Note: The two temporary positions of STORE are now not needed and another field may be moved over the two units positions used in the repetitive-addition method.

In this example, 7 to 9 storage positions (see note), and .3680 MS of processing time have been saved by using the repetitive addition method.

In all cases, multiplication by the repetitive-addition method for one-position multipliers will be faster than the multiply feature. In most cases of 2-position multipliers whose high order position is 3 or less, the repetitive-addition method will save time and storage.

Various multiply subroutines use considerably more core storage than the repetitive addition method and take substantially more time. However, practically speaking, the repetitive addition method can only be used for multiplication by constants. It is frequently practical to use a repetitive-addition multiplication subroutine, even if the multiply feature is available.

(H-3) Division Notes

1. The units position of the quotient is always located at the units position of the dividend field, minus the length of the divisor, minus 1, regardless of the number of extra decimal positions involved.

2. The dividend field must not have a B-bit in any position except its units position, where it may be required for sign control. A zone may have been developed in some position other than the units because of an arithmetic overflow in some preceeding program step. An improperly placed B-bit in the dividend field has the effect of reducing the value of the dividend, and consequently reducing the resulting value of the quotient. If the zone bits in other than the units positions of the dividend are either AB- or A-bits, they will be ignored.

3. If overflows are developed in the divisor, they are ignored.

4. In direct division, a zero divisor will signal a divide overflow. The original dividend will not be changed.

5. If there are not enough positions allowed for the quotient, the divide overflow indicator is NOT turned on, except when the divisor is zero. When the divisor is not zero, the divide op continues up to the capacity of the positions provided.

6. A word mark can appear anywhere in the dividend - quotient field positions. The divide op does not automatically generate a word mark in either the high order of the quotient, or the high order of the remainder.

(H-4) Addition during a Divide Op

Failure to clear the high order positions of the dividend field will result in the uncleared factor being added to positions of the developed quotient.

Under some circumstances, this may be desirable. If the developed quotient is to be added to another factor, this addition can be accomplished during divide. The factor must first be located in the correct digital registration in the quotient field. Some process time can be saved by this side-effect of the divide op.

Zone bits in the quotient positions of the dividend field will be removed. Therefore, the sign of the sum developed in the quotient positions must be the same as the developed quotient. The true arithmetic addition of the numeric value of any data in these positions will be added to the resultant quotient. Zone bits may be present in the units (sign) position of the dividend only.

(H-5) Division by Repetitive Addition

Instead of dividing by a fixed constant, if the reciprocal of the constant is used as the multiplier of either a standard multiplication using the multiply special feature or the repetitive-addition method of multiplication, the operation may be faster and require less core storage than use of the divide special feature. (See Multiplication By Repetitive Addition).

Listed below are some commonly used reciprocals.

Application	Actual divisor constant	Reciprocal constant used for multiplier
Inches to feet, units to dozens	12	.0833
Square inches to sq. ft.	144	.00694
Ounces to pounds	16	.0625
Minutes to hours	60	.0166
Feet to yards	3	.3333

Many other constant reciprocal factors can be used as the individual job requirements vary.

Note: This method of division is only practical when dividing by a constant. There also will be some loss of accuracy. Each of these techniques is more economical of time and core than either the multiply or divide subroutine. (Figure H-2)

APPLICATION	DIVIDE		MULTIPLY BY RECIPROCAL		
	SPEC. FEAT.	SUBROUTINE	SPECIAL FEATURE	REPETITIVE ADDITION	MULTIPLY SUBROUTINE
FEET TO YARDS (— 3)	1.2075 MS	26.411 MS	1.6905 MS	2.0125 MS	15.80 MS
MINUTES TO HRS. (— 60)	1.622 MS	26.730 MS	1.3585 MS	1.4835 MS	11.77 MS
SQ. IN. TO SQ. FT. (— 144)	1.6785 MS	27.054 MS	1.3700 MS	1.6905 MS	11.77 MS

Figure H-2. Comparative Timings of Division Methods

(I) Miscellaneous Operation Code Pointers

(I-1) Compare Instruction Chaining

In some limited applications, the compare op code may be chained. During I-cycles (instruction read-in time) of the compare op, the compare indicators are reset to equal (at I_2 time). A series of chained compare op's will not reset these indicators. Therefore, the composite compare answer will be available at the end of the series of compare op's. The first difference between a character in the A-field and the corresponding character in the B-field of the entire chained compare series will be the resultant answer. Once a compare indicator is set, it cannot be reset until the next compare op reaches I_2 time. Chained op's using the op-code only, never reach I_2 time.

This method of multiple field comparison (Figure I-1) can be used to advantage when several adjoining fields are to be compared with several other fields which are equal in length. Six core storage positions, as well as the process time required to read in these positions, is saved for every chained op.

```
Label     Op      Operands
COMP      C       AAREA, BAREA
          C
          C
          C
          C
          BE      EQUAL
          NSI
```

Figure I-1
Compare Chaining

(I-2) Compare Instruction used to Decrement Chained-Addresses

The compare operation code can be chained to decrement the A- and B-registers after a chained operation so that the registers are in the proper position for the next chained operation. The following example is used for simplicity: (For example, the length of the total fields could vary, etc.)

Label	Op	Operand	Comments
LOAD (Load the various edit words)			
EDIT	MCE	MI,0332	Edit first field.
	C		Correct A- and B-registers.
	MCE		Edit second field (chained).
	C		Correct A- and B-registers.
	MCE		Edit next field (chained).
	C		Correct A- and B-registers.
	MCE		Etc.
	C		
	MCE		
ROLL	A	MI, IN	Add minor field 1 to intermediate FLD 1.
	C		Correct A- and B-registers.
	A		Add minor field 2 to intermediate FLD 2.
	C		Correct A- and B-registers.
	A		Etc.

Label	Op	Operand	Comments
	C		
	A		
	C		
	A		
WRITE	W		Print edited line.
CLEAR	S	MI	Reset minor field 1.
	C		Correct A- and B-registers.
	S		Reset minor field 2.
	C		Correct A- and B-registers.
	S		Etc.
	C		
	S		
	C		
	S		
SWITCH (Branch to appropriate instruction)			

The core storage layout (Figure I-3) shows that spaces are left between each total field and the number of spaces will correspond to the decrementation accomplished by the compare operation. The space between each total field may be used for constants; so long as no additional word marks are inserted.

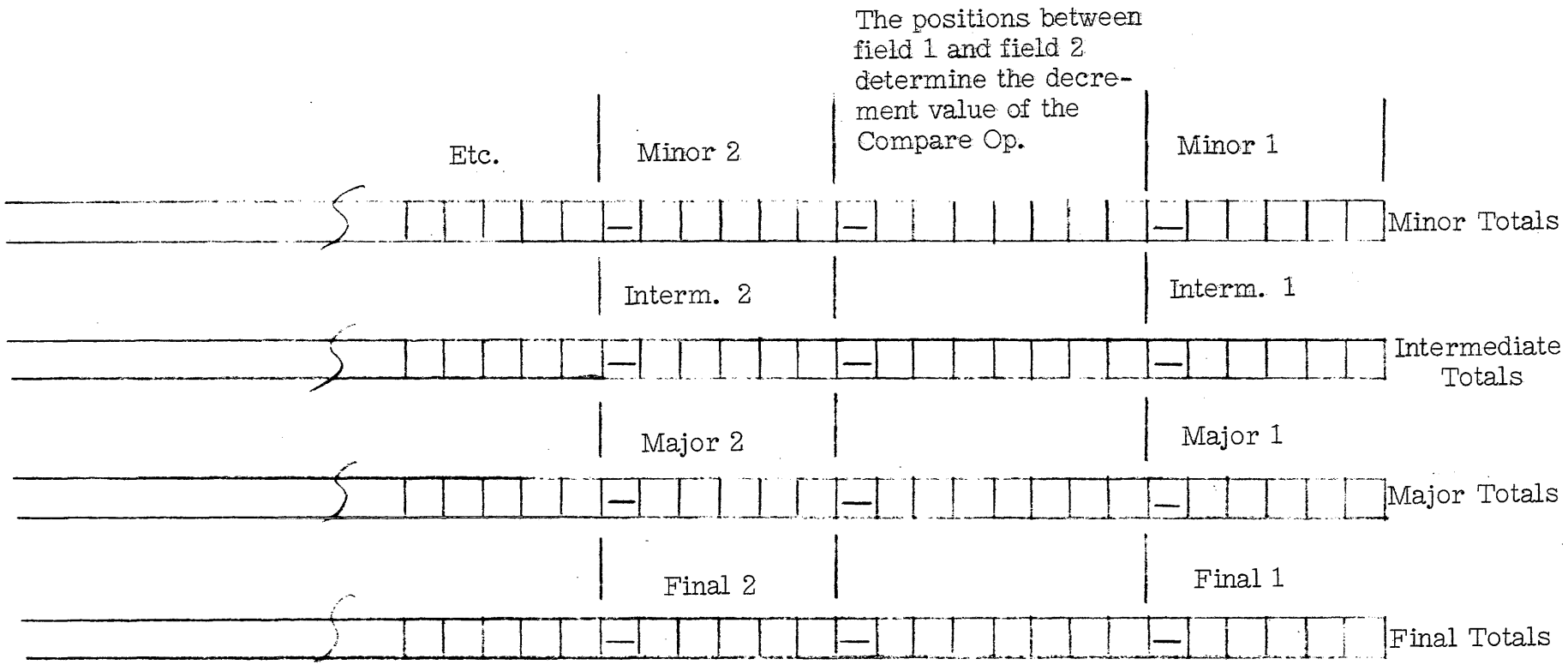


Figure I-3
Storage Layout for Compare Op Decrementing

This method is advantageous when several classes of totals are to be printed. The same routine is used, after insertion of the new addresses, in the EDIT, ROLL, and CLEAR instructions.

The reasons for conserving core storage must be weighed against the additional time required for this routine. If the routine is used several times (depending on number of fields, etc.), overall throughput can be substantially enhanced.

(I-3) Chaining Set and Clear Word Mark Instructions

When a string of word marks must be set or cleared, the instruction can be chained.

For example, if word marks must be set at locations 004 through 011 consecutively, the following chain will accomplish it:

Op	Operand	Comments
SW	7,11	Word marks in locations 7 and 11
SW		Word marks in locations 6 and 10
SW		Word marks in locations 5 and 9
SW		Word marks in locations 4 and 8

A similar chain of clear word marks will remove them.

If, in the same example, just the even numbered locations needed word marks, alternation of SW and CW would accomplish it:

Op	Operand	Comments
SW	6,10	Word Marks in location 6 and 10
CW		No word marks in location 5 and 9
SW		Word marks in location 4 and 8

(I-4) Data Movement without Setting Word Marks

When a numeric field is not defined by word marks, use the following method:

Op	Operand	
MN	KKK, LLL	
MN		
MN		
MN		(10 positions used)

rather than:

Op	Operand	
SW	KKK - 3	
MCW	KKK, LLL	
CW	KKK - 3	(15 positions used)

This method will save core storage for fields of 8 positions or less.

(I-5) Move, Load and Store Operations

An A-address can be entered into the A-address register without disturbing the contents of the B-STAR, when using move, load and store B-address register op codes as single address instructions. This permits the moving of non-adjacent A-fields into a string of adjacent B-fields, saving time and instruction storage space. The example shown in Figure I-4 gives a possible application, using the move op. The instructions for accomplishing this are:

Op	Operand
MCW	454, 311
MCW	
MCW	398
MCW	970
MCW	436
MCW	411
MCW	821
MCW	401
NSI	

Note that the first two fields to be moved were already adjacent, and required only the chained move op code.

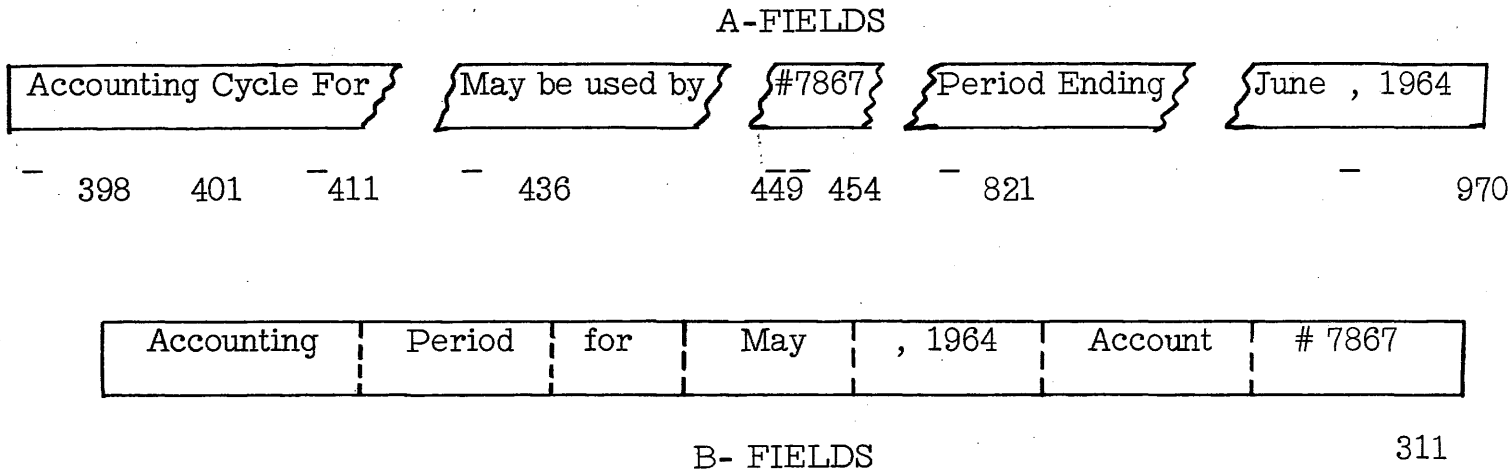


Figure I-4
Moving Non-Adjacent Fields into a String of Adjacent Fields

(I-6) Move Record Instruction

The move record op (MRCM, MCM or P op) instruction is terminated by the presence of a record mark (ǂ) or a group mark with a word mark (ǂ̄) in the A-field. This character is transferred to the B-field, except for the word mark associated with the group mark.

Thus, when moving data from a tape read-in area to the print area, the GMWM can be used instead of the record mark. (The GM will not print.)

The absence of a record mark or a group mark with a word mark may cause a move record op to blank out all, or a large portion of core storage.

(I-7) Move and Insert Zero Instruction

A group mark in the A-field will be moved to the B-field. A word mark associated with the GMWM is not transferred.

(I-8) Column Binary Operation

The move and binary decode instruction -- M AAA BBB A is terminated by a word mark in the high order of the A-field usually location 401. A group mark only will not stop this function. The group mark will be displaced. A word mark in either field will stop the operation, just as with the ordinary move (M) op.

(I-9) No Operation (NOP) Tips

An instruction can be no-op'ed provided the A- and/or B-addresses are valid, and the instruction length is either 1, 2, 4, 5, 7, 8 or more. (There must be a word mark between the no op instruction and the highest core location.) Thus, a NOP instruction can be longer than 8 storage positions, (such as N0123456789ABN) but it cannot be a length that is not other wise valid for other instructions (N 12345N).

Note that certain instructions can call on an index register inadvertently and the NOP A- or B-address can become invalid. If, in the instruction: N ICE A ---, index register 3 contains a factor greater than 064, an invalid effective address will be developed in the A-register (ICE = 15,935 + index register 3. If index register 3 is more than 064: 15, 935 + 100 = 16,035 will be developed.)

(I-10) NOP of I/O Instructions

If it is necessary to NOP an I/O op, such as tape, disk, etc., the unit control designation must also be changed to N. For example, L%B6 234 R will become NNB6234R. Notice that a word mark is not necessarily associated with the unit-control position of the instruction.

(I-11) Edit Instruction Pointers

1. Zero and punctuation suppression can be reinitiated by any alphabetic or special character, except the punctuation marks: comma, decimal, and hyphen. Any other character will cause suppression to be reinstated during editing with zero suppression on a system with the expanded print edit feature.

2. Any sign (zone bits) in the units position of the data word (A-field) is removed by the edit instruction. If this sign is required for subsequent program steps, it must be stored in another core location before the edit instruction is executed and replaced after the edit. The sign will be used by the edit instruction, but it will not be regenerated.

3. Any valid character used in the edit control word will be regenerated in the control word storage locations, and can be used again without modification.

4. Although floating-dollar and asterisk-protection cannot be used in the same edit word, a dollar sign can be placed to the left of the asterisks by inserting it in the edit control word so that there is at least one blank position separating the dollar sign and the asterisk.

   Thus, 12345 edited into $bbb, b*0. bbCR* becomes $****123. 45. The factor 0000N becomes $*******. 05CR* after an edit using the above control word.

5. Some examples of control words and their results when the data word is zero:

bbb, b$0. bb	$. 00
bbb, b*0. bb	*******. 00
bbb, bbb. $0	bbbbbbbbbb
bbb, bbb. *0	**********

(I-12) Store B-Address Register (SBR) Instruction Pointers

1. Indexing bits in the ten's position of the B-address are not stored. If indexing bits are required, they can be transferred using the move zone op code. Thus:

```
Label    Op     Operand
MOVE     MCW    XXX, 6X8
         SBR    901
         MZ     MOVE + 5, 900
```

   However, if the affected address has been modified by indexing, the resultant address will reflect this modification. Thus, if index register one contains the factor 010, and the B-field length of the input area is 20 positions, the instructions:

```
MU %U3 5T3 R
SBR 321
```

   will cause the address 564 to be stored at location 321. Note that in this particular op, the address which is stored is actually one position past the GMWM. The indexing of the address 5T3 gives an effective address of 543. The field length of 20 positions gives an effective ending address of 562 (actually 563 because of the GMWM). The stored address, then, becomes 564. This address factor will not reflect the original address' indexing.

2. An address constant, or any 3-position constant, can be stored by using the SBR (H) op: N 444, SBR 635. The constant 444 will be stored in location 635. Note, however, that had the NOP A-address been indexed, the indexing bits would NOT have been stored: N 5V5, SBR 666., The address 555 will be stored at location 666, and the indexing bits will have been lost.

   The following technique will cause the constant 999 to be stored at location 888 and one core position will be saved over the previous method. SBR 888, 999

3. A SBR op cannot follow any conditional branch instruction (or any command having other than 4, 7 or 8 positions). During the I-phase of the conditional branch instruction, the d-character is read into the hundreds position of the B-STAR (or A-STAR for a 2-position op), and blanks are placed in the tens and units positions of these registers. If the branch is not successful, the SBR op will store an invalid address. In the following example, both the hundreds and thousands positions of the B-storage address register are loaded, but the tens and units positions are left blank:

```
C   AAA, BBB
B   III, S
SBR AAA
```

In this case, the B-STAR to be stored, if the compare is not equal, will be invalid (12bb) and subsequent execution of this routine will produce an invalid address error.

4. Any three storage positions can be reset to zero or any other 3-character factor (without regard to word marks) as follows:

```
SBR   FIELD, 0        or
SBR   FIELD, 555
```

In this example, the label FIELD refers to the right-most of the three positions being reset. This procedure can be used to reset 3 independent 1- position counters or index registers. An ADCON (autocoder) or a DSA (SPS) both requiring 3 extra positions are not required for the address constant: 000.

5. The following instruction can be used to increment or decrement an index register on systems equipped with the store A- and B-address register features.

```
SBR   X1, A + X1
```

where A is the value to be added or subtracted from the index register. For example:


```
SBR   89, 15 + X1      Index register 1 is incremented
                       by 15.

SBR   89, 15 998* + X1 Index register 1 is
                       decremented by 2.
```

6. To save the contents of an index register and restore it at the same time, use:

```
Label      Op      Operand
           SBR     RESTOR + 6, 0+X1
           ....
RESTOR     SBR     X1, 0
```

7. As a useful program linkage, the following places the appropriate address in a common routine.

```
Label      Op      Operand
           SBR     INSTR +3, FIELD
INSTR      MCW     0, GO
```

* Core size dictates whether this method of decrementing is valid.

(I-13) Subroutine Linkage with and without Store B-Address Register Instruction

It is very economical in terms of saving core storage to be able to provide linkages to closed common subroutines. Using the SBR instruction simplifies this task, as follows:

Label	Op Code	Operands	Remarks
	MCW	VALUE, AREA1	Set up values for subroutine or
	MCW	NUMBER, AREA2	macro instruction
	B	SUB	Branch to subroutine

SUB	SBR	ENDSUB + 3	Store NSI in last instruction of subroutine
	-		
	-		
ENDSUB	B	0	Branch back to main line

If the machine in question does not have the SBR special feature, the following routine can provide the same linkages:

Label	Op	Operand	Remarks
	MCW	VALUE, AREA1	Set up values for subroutine or
	MCW	NUMBER, AREA2	macro instruction
	MCW	SBR + 3, ENDSUB + 3	Move branch instruction to return linkage
	B	SUB	Branch to subroutine

Label	Op	Operand	Remarks
SBR	B	MAIN	Branch linkage constant
MAIN	A	FIELD, AREA3	Main line processing
SUB	-		Begin subroutine
	-		
ENDSUB	B	0	Branch back to main line

The same routine can be more simply stated with fewer labels:

Label	Op Code	Operand	Remarks
	MCW	VALUE, AREA1	Setup subroutine or
	MCW	NUMBER, AREA2	macro values
	MCW	* + 8, ENDSUB + 3	Setup return branch
	B	SUB	Branch to subroutine
	B	* + 1	Branch linkage constant
	A	FIELD, AREA3	Main line processing
	-		
	-		
	-		
SUB	-		Begin subroutine
	-		
ENDSUB	B	0	Branch back to subroutine

(I-14) Store A-Address Register (SAR) Instruction Pointers

1. If the A-address of a function is required in more than one location, use:

```
A       AAA     BBB
SAR     789
SBR     987
SBR     654
etc.
```

At the end of the SAR op, the B-STAR will have the address previously contained in the A-STAR. Therefore, an SAR op can be followed by as many SBR op's as might be required to satisfy the program.

Storing both the A- and B-addresses is less convenient. Example 5 of this section shows one technique to do this.

2. The primary use of SAR is in deblocking input records. When the blocked records are separated by a record mark, the following routine will keep track of the address of the next record without having to otherwise increment the index register:

Label	Op Code	Operand	Remarks
GO	RT	1, MASTER	Read tape
	BER	RDERR	
	BEF	REOF	
	SBR	X1, MASTER	Place addr. of first record in ind. 1
	BCE	GO, 0+X1, ≢	Fully processed?
	MRCM	0+X1, PROCES	Logical record to work area
	SAR	X1	

3. Another deblocking technique, without using index registers, enables the programmer to change the blocking by changing only the DA statement associated with the file.

Label	Op	Operands
EOBTST	BCE	READ, 0, ≢
MVSTEP	MRCM	INPUT, WORK
	SAR	MVSTEP+3 (To save the next A-addr.)
	SBR	EOBTST+6 (To save A-addr again)

4. SAR can be used in routines to reset an area to blanks or fill an area with any character.

a. If the area to be cleared contains a word mark, only in the high order position, SAR is not needed. For example:

Label	Op	Operands	
WKAREA	DA	1X100	
FIRST		1	
LAST		100	
	-		
	MCW	@ @; LAST	
	MCW	LAST	Moves blank to last -1 and continues until high order W/M is sensed.

b. If the area contains many fields with multiple word marks:

Label	Op	Operands
WKAREA	DA	1X100
FIRST		1, 5
SECOND		7, 9
THIRD		11, 14
-		
-		

Label	Op	Operand
TWENTY		90,97
LAST		98,100
	-	
	MCE	@ @, LAST
	SBR	X3 save last -1
HERE	MCW	1+X3, 0+X3
	SAR	X3 save addr of unblanked position
	C	X3, LIMIT
	BE	MAIN to main routine
	B	HERE
LIMIT	DSA	FIRST -1

Note: Whenever a word mark is sensed, an extra move instruction is given.

5. SAR is useful when transferring data in one area to another, where differences in word mark configurations present a difficulty, e. g., work area to master output area.

Label	Op	Operand
HERE	MCW	MOVE + 6, TO MOVE + 6
	C	MOVE + 3, LIMIT
	BE	MAIN to main routine
MOVE	MCW	A FIELD, B FIELD units position of each area
	SAR	MOVE + 3 to save next A-addr
TO MOVE	MCW	0, 0
	SBR	MOVE + 6 to save next B-addr
	B	HERE
LIMIT	DSA	XXX high order of A-field

Note: Although each move instruction is executed twice, this technique saves considerable core storage.

(I-15) Notes on Index Register Timing

Indexed instructions require three to four additional I-cycles per indexed address and, therefore, additional instruction time. For example, moving ten characters requires (7 + 1 + 20) 11. 5 us or 322 us without indexing. Indexing of one address adds 34. 5 us, two addresses, 69. 0 us.

While indexing easily outperforms address modification, one case where indexing may not be the best method is that of indexing fields in a tape I/O area. In this case, every logic instruction referring to I/O data is indexed and time may be greatly increased. The use of a work area would reduce the increase in process time. This alternative will use more core, however, and should be weighed accordingly.

Indexing time is especially worth considering if iterative routines (e. g. programmed multiplication) are a basic part of the program logic or if a slight increase in process time may cause an interlock in another I/O device such as the reader or punch.

(I-16) Testing for an Odd Character

A. Method I (if column binary feature is available)

```
BBE     XXX, YYY, 1
where   XXX = address if character is odd
        YYY = address of character tested.
```

Positions required: 8

B. Method II Step-by-step testing

```
BCE  XXX, YYY, 1
BCE  XXX, YYY, 3
BCE  XXX, YYY, 5
BCE  XXX, YYY, 7
BCE  XXX, YYY, 9
```

Positions required: 40

C. Method III Chained Testing

```
MCW   YYY, * + 8
BCE   XXX, CONST, ?
BCE
BCE
BCE
BCE
-
CONST DCW  @13579@
```

Positions required: 24

(J) Magnetic Tape Considerations

(J-1) Clearing Group Marks from Tape Readin/Readout Areas

Tape information can be lost if a spurious group mark with a word mark (GMWM) is generated within the tape read (or tape read/write) area. The tape read-in and write-out areas should be cleared after each read or write operation, respectively.

Tape read is terminated when a GMWM is sensed in storage. The last character accepted will be one position to the left of the GMWM. If the tape read operation is terminated by an interrecord gap (IRG), a group mark without WM is written one storage position past the last data character written into storage. For fixed-length records, this GM will normally fall over the existing GMWM. For variable-length records, this GM may fall anywhere in the read-in area.

A GM without WM may be a character of the tape record. This may be caused by a read parity error, or may be a program requirement.

If the GM happens to fall over a WM, and this GMWM is not cleared, subsequent records will be effectively ended at this new GMWM.

Program around this condition by using a Store B-Register op and two chained Clear Word Mark ops. These instructions must follow the tape read op without intervening steps, as shown in Figure J-2.

LABEL	Op	A	B	d	COMMENTS
RDTAPE	MU	%Ux	BBB	R	Read tape record into BBB.
	SBR	CLRWM+3			Store B-STAR; Address of GM + 1.
CLRWM	CW	000			Clears WM at GM + 1
	CW				Clears WM under GM
	NSI				

Figure J-2 Subroutine to clear word mark under group mark.

In the program steps preceding the next tape read Op, a word mark may have to be placed under the correct GM, and, if needed, at GM+1.

Another way to do the same thing, without clearing the WM at GM+1 is as follows:

LABEL	Op	A	B	d	COMMENTS
RDTAPE	MU	%Ux	BBB	R	Read tape record into BBB
	SBR	X1			Store B-STAR in Ind. Reg 1; Address of GM+1.
	MN	0+X1			Moves number portion of GM+1 into GM+1 but steps down A-Register to address of GM.
	CW				Clears WM under GM.

Both of these routines may be used with SW to set a work mark under the GM.

(J- 2) Compressed Tape Instruction Used to Read Regular Records

This command (special feature) is terminated only by an IRG. It is not stopped by a GMWM in core. Therefore, this instruction can be used whenever there may be spurious GMWM's in the tape read-in area. It is not necessary to clear the GMWM's. Note, however, that since the operation is not halted by a GMWM in core, an extra-long tape record could wipe out core storage beyond the tape read-in area. A group mark without a word mark is inserted in core when the IRG is finally sensed, as in normal tape read. However, the B-STAR will contain the address of the GM plus 1, as in normal tape read.

(J - 3) Tape Record De-Blocking Routine (See Figure J - 1)

To pick off each record of a blocked set of variable-length records, and find the end of the block as a by-product of the basic operation, use the following program format:

```
LABEL   OP    OPERAND              COMMENTS
        SBR   MOVE +3,INPUT        Initialize Move Record A-Address
MOVE    MCM   INPUT,  WORK         Record mark or GMWM stops move.
        SAR   MOVE+3               Store A-STAR for next move.
        SBR   END + 6              Store old A-STAR (by using the
                                   SBR Op) for GMWM test.
END     BCE   READ,  000,          Test for GMWM.  If yes: Read tape.
                                                   If no:  Move next record.
        B     MOVE
```

Representation of a section of magnetic tape.

Tape record # 1	Record # 2	Tape record number 3	I R G

Representation of a section of core storage.

Tape record # 1 in storage	Tape record # 2	Tape record # 3

Figure J - 1 Tape Record De-Blocking

(J- 4) Diagnostic Tape Read Instruction

A little known tape instruction is the diagnostic tape read:

```
CU   %UX, A
```

This instruction allows a tape record to be read, checked for tape validity, tested for end of file, but does not enter the data into storage.

The instruction has at least 3 possible uses:

1. Pass a tape and bypass a predetermined number of tape records.

2. Pass a tape and bypass a predetermined number of tape files, on the same reel, separated by tape marks.

3. Check a tape for validity as a multiprogrammed operation during normal running. This is especially useful when a tape is to be read that was written on a tape drive without a dual-gap head or dual level sensing.

After a diagnostic tape read is executed, the processor is immediately released for other instructions. The IRG stops the read and the diagnostic tape read must then be executed. The EOR (end of reel) indicator in the tape drive will be turned on if a tape mark is read, but not when the end-of-reel reflective sticker is sensed, since the tape drive will be effectively in read mode.

Other tape operations are interlocked until the check character for the record being bypassed has been read.

The Diagnostic Tape Read op does not interlock the CPU. Processing continues. If this op is being used to determine the existence of a tape mark, and the program depends on recognition of this character, the system must be interlocked. This can be accomplished by coding a test-for-tape-error just prior to the point in the program dependent on the tape mark condition. The Branch if Tape Error instruction cannot be executed until the IRG is reached.

The Tape Error indicator will be turned on if the record being bypassed was not in the parity dictated by the A-address of the diagnostic tape op. The letter B in the ten's position of the A-address of the tape instruction specifies odd tape parity, while the letter U specifies even tape parity. Note: do not confuse tape parity with processing unit parity.

(J-5) Miscellaneous Tape Notes

1. Tape Load Operation using Tape Load Key

   A GM/WM in core will not stop a tape-load operation when initiated from the tape-load key. This operation stops only when an IRG is sensed. Any characters (including a group mark with a word mark) will be erased during the read portion of a tape-load.

2. Tape Read Instruction

   A missing group mark with a word mark at the end of a tape read-in area can cause a large portion of core storage to be blanked out. The tape op will be terminated in this case by the IRG.

3. Tape Rewind

   Tape unit rewinding should be included in the housekeeping routine. This insures that all tapes being used for the job will begin at load point.

4. Addressing GM/WM when Writing Tape

   If a tape-write instruction is given to write a GM/WM only, the tape unit will create an IRG of 1 1/2 inches instead of the usual 3/4 inch gap. The tape error latch will be turned on.

   A backspace command at this point in the program will cause the tape unit to back up over the wide IRG as well as the last record written (not the GM/WM, which does not go on tape).

   Note: Depending upon the series of the system, the above operation may cause the entire system to interlock, and will require that the start reset key be pressed.

5. In order to assure that a valid tape mark has been written on tape, use the following routine:

Label	Op	Operand
WRTTM	WTM	3
	BSP	3
	RT	3, INPUT
	BEF	EOJ
	B	WRTTM

(J-6) Noise Records

A noise record may be read into memory. There are several approaches to determining if a noise record has been read.

1. Since a noise record can be 1 to 13 characters, plan each tape file so that no valid tape record is less than 14 characters.

2. Use a string of special characters in positions 1-14 of the tape read in area. After a read, check for their presence to indicate a noise record.

3. Clear the entire noise area after a noise record is found.

4. Set a GMWM one position past where the GM should fall in a read tape area. Leave the GM position blank. After a read, check for the GM as a check for wrong length (long or short) tape record. Blank out the generated GM after a good tape read.

(J-7) Read Tape Mark Effect

When a tape mark is read, it not only gives an EOR indication, but also reads into storage. It is followed by a group mark. This GM may read in on top of an existing word mark, creating a GMWM. Any End-of-File routine must include a tape read-in area-clear routine as illustrated in a previous section. Reset any word marks for subsequent tape read operations, if required.

Note: On a tape-error transmission, parity error constituting a group mark may be left in storage following the read op. It is advisable to clear the read area to eliminate this group mark, before the next read try is attempted. This operation will use process time during which the machine is waiting for the TU to complete a backspace, so no actual time will be lost.

(J-8) Skip Tape and Erase Instruction

This instruction -- SKP 3, is effective for the next tape write op or tape mark write op. The latch is not reset by a rewind, rewind-unload, or backspace command. Use care to insure that this instruction is executed before the tape is backspaced or rewound. Otherwise, the first write command for that drive after the backspace or rewind will cause a skip and erase to take place. This could cause difficulty on the next use of that tape drive, if SKP is used at end of job.

Ordinarily, this is of little consequence since the automatic load-point skip will take place anyway.

(J-9) Tape Transmission Error and End of Reel Indicators

1. There is only one tape error latch. It is set on if a tape read or tape write error occurs on any tape drive. This latch can be tested and reset by the Branch if Tape Error instruction -- BER III. The tape error latch is also reset at the beginning of any tape command for any tape drive. If the tape error is not tested before the next tape op, the error condition is lost.

2. Each tape drive has its own End of Reel latch. This latch is set on when either an end-of-reel sticker (reflective spot) is sensed while writing, or when a tape mark character (BCD code 8, 4, 2, 1) is read as the first character of a tape record. This latch can be reset by the Branch if End of File or Reel instruction -- BEF III, or by the manual unload push button on the particular drive having the end-of-reel condition. (The Tape Indicate light will be lighted on this drive.)

   A tape drive must be in Select and Ready status to allow a test for end of reel. Therefore, programming caution must be used to insure that an EOR test is associated with the proper tape drive. Any tape command referring to a specific drive places that drive in select status. Ready status infers that tape is loaded in the drive, and the unit is otherwise physically ready (ready light is on).

   The end of reel indicator will be turned on in a tape unit when either a reflective sticker is encountered during a tape write op, or when a tape mark (BCD code 8, 4, 2, 1) is read as the first character of a tape record. This means that any multi-character record having a tape mark as its first character, even though this record may be a noise record, will turn on the end of reel indicator.

   Therefore, it is important to test for a tape transmission error before testing for an end of reel condition. Note, however, that the EOR indicator is only reset by the Branch if End of Reel instruction, and not by the next tape op, as is the case with the tape error latch. Therefore the test for EOR must be made while this drive is still selected. This is usually accomplished by branching back to the EOR test following the completion of the tape error subroutine, or if no error existed, going directly to the EOR test which follows the tape error test instruction:

```
Label      Op      Operand
           RT      3, INPUT
           BER     XXX
EORTES     BEF     YYY
           NSI
           .
           .
           .
XXX        -
           -
           -
           -
           B       EORTES
```

(J-10) Trailer Nines Records for End of File Recognition

The use of a trailer record with a control field consisting of all nines for each input tape file will generally eliminate the need for special end of file switches in the various comparison routines. This nines record may either be read in from tape or may be generated in core storage by the program when an end of file condition is recognized.

(J-11) Operation of 729 Tape Drives

Tape Mounting

1. Allow about three feet of magnetic tape to hang freely from the reel that you are going to mount. Place reel on drive and press firmly to insure that the reel is properly seated. Tighten the reel on the drive by turning the knob clockwise until it becomes difficult to turn.

2. Thread the magnetic tape through the guides and rollers insuring that the glossy side of the tape is up when passing under the read-write head and that the tape is not twisted.

3. Place the end of the tape on the right reel, hold the reel release button down, and wind the tape on the reel until the reflective spot passes under and to the right of the read-write head.

4. Close the front glass door and press the keys in the following order:

   a. Reset
   b. Load Rewind
   c. Start

5. Insure that you have the proper density setting.

Tape Unloading

1. Press the following keys:

   a. Reset
   b. Load Rewind

2. After the tape has rewound, press:

   a. Unload

3. Depress the reel release button and manually rewind the tape on the left reel. Loosen the knob and remove reel.

(J-12) Operation of 7330 and 7335 Tape Drives

1. The 7330 and 7335 read-write head must be lowered manually by the use of a black plastic handle located at the head.

2. The 7330 and 7335 tape drive should be prepared for operation using the following sequence of steps:

   a. Open center cover and right column door.

   b. Thread the tape through the guides and rollers.

   c. Wind tape around the take-up reel; use reel release button.

   d. Wind load point past head. Leave no slack in the tape.

   e. Close the doors on the horizontal vacuum columns.

   f. Lower the read-write head with the reel release button depressed.

   g. Release reel release button until vacuum comes up.

   h. Depress reel release button and turn left reel clockwise and right reel counter-clockwise to load tape in columns.

   i. Close door.

   j. Press reset, low speed rewind and start buttons.

   If the tape is loaded into the vacuum columns before the doors on the columns are closed, the tape may be pinched.

3. Make sure that all tape drives are unloaded (read-write head up) before turning off power to the system. If this is not done, blown fuses may result.

4. The 7330 and 7335 is sensitive to the lateral positioning of the aluminun load point strip. The end of reel strip is positioned toward the rear of the width of the tape. If the load point strip is not positioned forward enough, the 7330 and 7335 will recognize it as end of reel indication and cause the appropriate indicator to be turned on.

5. It should be noted once again that a "rewind" instruction causes a low speed rewind. "Rewind and Unload" causes a high speed rewind. The choice of the two must depend on program requirements. If "Unload" is used, to reload the tape must be reloaded into the vacuum columns, the read-write head lowered and the appropriate buttons reset.

6. It is possible to improperly thread the tape on these drives and, outwardly, the drives operate properly. If the tape is read back with the tape still incorrectly threaded, it will work well. However, the tape cannot be read again on a 729, 7330, or 7335 with the tape threaded properly.

(J-13) Tape Handling

Magnetic tape must be protected from dust and dirt; foreign particles can reduce the intensity of reading and recording pulses by increasing the gap between the tape and the head.

1. Keep tape in a dust-proof container whenever it is not in use on a tape unit. During loading, take the tape directly from the container; after unloading, place the tape directly in the container.

2. While the tape is on the machine, keep the container closed and put it where it is not exposed to dust or dirt.

3. Store tapes in an elevated cabinet away from paper or card dust to minimize the transfer of dust from the outside of the containers to the reel during loading or unloading.

4. Do not use the top of the tape unit as a working area. Placing material on top of the units exposes it to heat and dust from the blowers and may interfere with cooling the tape unit.

5. When identifying tape reels, use a material that can be removed without leaving a residue. Adhesive stickers, easily applied and removed, are satisfactory. They can be prepared in advance and applied during the loading procedure. Never alter identification by changing labels with an eraser.

6. Place load points and reflective spot on tapes with care. Properly align and press them tightly on the tape with the back of the fingernail, preferably while the tape is loaded on a unit. If it is done away from the unit, keep unrolled tape off the floor and away from dust.

7. Inspect containers periodically; remove accumulated dust by washing with a household detergent.

8. When necessary to clean tape, wipe it gently with a clean, lint-free cloth moistened with IBM tape transport cleaner. Do not do this with H. D. tape.

9. Exercise extreme care when removing the file protection ring. Under no circumstances should the ring be removed while the tape is loaded in the columns.

Recorded information comes within .020" of the edge of the tape. Proper operation relies on the edges being free from nicks and kinks.

1. Reels should be handled near the hub whenever possible. If a reel is difficult to remove, break the bond between the reel and the hub by placing the palm of the hands on the periphery of the reel and rotating it. Never rock the reel by grasping the outer edge.

2. Carefully avoid pinching reels or contacting the exposed edge of the tape.

3. When installing the reels, push them firmly against the stop on the mounting hub to insure good alignment.

4. Take special precautions to be sure the hub is tightened after the reel has been mounted.

5. When placing the tape on the take-up reel, carefully align the tape to prevent damaging the edge on the first few turns.

6. When winding the tape to load point, rotate the machine reel with the finger near the hub and on the reel. Rotating the reel with the finger in the cut out, nicks or curls the guiding edge of the tape.

7. Always place sponge rubber grommets or special clips on stored reels to prevent the free end from unwinding in the container.

8. If tape breaks, divide the reel into two smaller reels. Splicing is not recommended. If necessary to make a temporary splice to recover information, be sure to use special low cold flow splicing tape (Customer Engineering supply item).

9. Dropping a reel can easily damage both reel and tape. Use of a reel and tape after they have been dropped is usually unsatisfactory.

10. Never throw or mishandle reels, even while they are protected in their containers.

11. Allow the tape unit to complete the unload sequence before opening the door.

Magnetic tape, especially acetate tape, is sensitive to changes in humidity and temperature. Take the following precautions:

1. If possible, store tape where it is to be used (in the computer room). Tape storage near the tape units reduces handling and variations in atmospheric conditions.

2. The atmosphere should be controlled between the following limits:

Relative humidity	40% to 60%
Temperature	65° to 80° F

3. If tape must be removed from the computer room atmosphere, hermetically seal it in a plastic bag. If tape is not hermetically sealed then, it must be returned before reuse and allowed to remain in the computer room atmosphere for a time equal to the time it was away from the room. Twenty-four hour conditioning is necessary if the tape was removed for more than 24 hours.

When shipping magnetic tape, the following procedure is advisable:

1. Pack the tape and reel securely in a dust proof container.

2. Hermetically seal the container in a plastic bag. (Ordinary plastic bags that can be sealed with a hot iron should be available from local merchants.)

3. Obtain additional support by enclosing containers in individual stiff cardboard shipping boxes.

For long-term storage, take the following precautions:

1. Provide proper mechanical support for the reels by using the dust proof containers.

2. Enclose the reel and container in a hermetically-sealed moisture-proof plastic bag.

3. Store tape in an area of constant temperature (between 40 and 120° F is satisfactory). Either freezing or excessively hot temperatures could harm the tape.

If a tape reel is dropped, the reel may be broken or bent, the edge of the magnetic tape itself may be crimped, and the magnetic tape may be soiled.

The tape should immediately be inspected. Breaking or bending can usually be verified by visual inspection. Bending can also be verified by mounting the reel on the hub of the tape frame. If the reel is bent or broken, it should not be used; the magnetic tape, however, may be serviceable and can be wound on another reel.

If the edge of the tape is crimped, steps to be taken depend on whether it contains essential information. If the tape contains no essential information, discard the footage with the crimped edge. If the tape contains important information, reconstruct it through tape-to-printer or other machine operation. If this fails, the records in question must be recreated from the original input or control data.

Any time a tape reel has been dropped, clean the tape and reel thoroughly.

If visual inspection fails to uncover any evidence of breaking or bending of the reel, or crimping or other damage to the magnetic tape, assume that the tape is in good operating condition. If possible, make a test to verify that the tape operates properly before using it on subsequent runs.

The following are points of general tape-handling information:

1. Senior operators should always take special precautions to follow the tape handling recommendations to show, by example, the care required to insure good performance.

2. Replace any tape arriving at the customer's installation in unusable condition and return the faulty tape to the factory. To aid the factory in its inspection, ship the tape according to the shipping instructions outlined in this section.

3. Use discretion about smoking in the vicinity of tape because smoking adds to the dirt problem. Also, a hot ash could cause serious trouble with a reel of tape.

Mylar magnetic tape should be handled in the same way as acetate tape. However, if Mylar tape is removed from the computer room atmosphere for short periods (not in excess of 3 months), it is not necessary to hermetically seal the tape nor to recondition it after return to the computer room atmosphere. For long-term storage, Mylar tape should be hermetically sealed to guard against dirt, dust, and excessive moisture.

WARNING: Never store reels of tape near magnetic fields.

(J-14) Tape-Transport Cleaning for 729 Tape Drives

The tape drive transport mechanism should be cleaned at least once every eight hours, or every ten full reel passes, whichever occurs first.

The materials required for cleaning the transport are available in a tape drive cleaning kit, P/N 352465. DANGER. Caution should be exercised whenever the transport cleaner is used.

Prolonged or repeated contact of the tape transport cleaner with the user's skin should be avoided.

Split Guides

Use the brush and thoroughly remove all oxide accumulation on the surface and between the two ceramic elements.

"H" Shield

The underside of the "H" feed-thru shield should be cleaned with a lint- free cloth or pad mositened with the approved cleaning fluid.

Rewind Idler Pulley

Clean with a lint-free cloth or pad moistened with the approved cleaning fluid.

Drive Capstan

Do not clean the drive capstan while it is rotating under power. Use the brush handle wrapped with the cleaning cloth and scrub vigorously. The capstan must be rotated manually.

Nylon Pulley

Use a lint-free cloth or pad and the approved cleaning fluid. A motion around the circumference of the pulley should be used. Do not rub too hard in any one spot.

Stop Capstan

Use a lint-free cloth or pad moistened with the approved cleaning fluid to clean this item. Do not rub where the nylon pulley contacts it.

Cleaner Blade

Use a lint-free cloth or pad moistened with the approved cleaning fluid to clean this area. Do not rub hard on the cleaner blade.

Read/Write Head

Use a lint-free cloth or pad moistened with the approved cleaning fluid to clean the head. Scrub in the direction of tape movement, never across the head.

Vacuum Columns

The columns should be cleaned weekly with the approved cleaning fluid. DO NOT, under any circumstances, use any metal instruments to clean the columns. Frequency of cleaning may need to be changed, depending on the type of tape and the amount of tape passed.

Cleaning the transport area should be done using a minimum amount of cleaning fluid. The cleaning cloth or pad should be damp and not saturated with cleaning fluid when cleaning. Occasionally, loose fibers will detach from the cleaning cloth or applicators during cleaning. A visual inspection should be made to be certain that none of these loose fibers remain in the transport area after cleaning.