SOAP
IBM 650 Symbolic Optimal Assembly Program

Programmer's Guide

Prepared by
Edna Vienop

March, 1957

TABLE OF CONTENTS

Introduction

The purpose of this report is to describe coding procedures and machine operating rules for "SOAP", the N. Y. Symbolic Optimal Assembly Program for the IBM Type 650 Magnetic Drum Calculator.* The main features of this system are:

1) Ease and speed of programming since most location and instruction addresses may be left blank.

2) Five-character "free" symbolic addresses.

3) Ability to restrict the assembled program to any part of the drum.

4) Assembly of most programs in one pass.

5) Simultaneous assembly and optimization.

6) No restriction on the total number of symbolic addresses.

7) Incorporation of either relocatable absolute or symbolic library programs.

Definition of Terms

L = location (address)
D = data (address)
I = instruction (address)
FWA = first word address
LWA = last word address
Δ = number of words in a block
OP = operation
block = a consecutive group of drum locations
region = a block designated by an alphabetic character
pseudo-instruction = an instruction which is never executed
symbolizer part = left most position of the L, OP, D or I field
absolute part = L, OP, D or I field with symbolizer part deleted

Coding Form and Input Card Format

Figure 2 shows the coding form to be used when writing for SOAP. Vertical dotted lines separate the symbolizer and absolute parts of the L, OP, D and I field. Observe that the input card format is indicated at the top of the coding form. Punching in columns 1-39 is immaterial. Column 40 must contain an x punch.

Instructions and data enter the assembly one word per card. The output is the standard one word per card.

*equipped with an alphabetic device, selectors totaling 20 pilot selectors, 20 co-selectors, and 5 punch code selectors.

Specifying Program Area on Drum

Prior to assembly, the entire drum is considered as available to the program. It is usually necessary, however, to prevent the program from occupying certain drum locations, e. g., read and punch areas, data areas, etc. There are two special types of cards which can be used to restrict the program to any predesignated part of the drum:

Block Reservation Card: Type 1
This card contains two absolute addresses called FWA, Δ punched in the absolute part of the D and I fields, respectively. All locations from FWA to LWA (inclusive) are made unavailable to the program, where LWA = FWA + Δ -1.

Regional Specification Card: Type 3
This card contains an alphabetic character to be associated with a region on the drum defined by the addresses FWA, Δ. As in the case of a block reservation card, all locations from FWA to LWA (LWA = FWA + Δ - 1) are made unavailable to the program. If the programmer wishes to refer to locations within a region using "regional" addresses (see Regional Addresses, page 3), a regional specification card defining this region must precede the program being assembled.

Format for Type 1 and 3 Cards:
The format for the preceding two types of cards is shown in Figure 2. The type number must be punched in the column so labeled. The symbolizer part of the D and I fields must be blank. The symbolizer part of the location of regional specification cards (* in Figure 2) contains the alphabetic character associated with the region.

Col	40	41	42 43 → 46	47 48 49	50 51 → 54	55	56 57 → 60	61	62	63 → 72
	x	TYPE	Location	Oper.	Data Address	TAG	Inst. Address	TAG	SIGN	Remarks
	x	1			←FWA→		←Δ→			Block Reservation
	x	3	*		←FWA→		←Δ→			Regional Specification

Figure 2. Format for Type 1 and 3 cards

Instruction Format

The instruction format for SOAP is as follows:

L	OP	D	T	I	T
X|XXXX	X|XX	X|XXXX	X	X|XXXX	X

The various types of addresses and operation codes are described in the following sections.

Types of Addresses (L, D, or I)

Regional Address

A regional address has the form

"A"	NNNN

where "A" is any alphabetic character and NNNN is a four digit number. In order to utilize regional addresses, one must first define a region by a regional specification card. It is possible to have as many as twenty six different regions (one for each letter of the alphabet). The regional address of the first word of region B, for example, is B0000. The twelfth word of region Q has the regional address Q0011.

One need not employ regional addresses for a region whose limits are known. The most important use of regional addresses is to facilitate addressing words within a block whose terminal addresses will not be absolutely determined until completion of the program.

Regional addresses usually refer to blocks of data.

Symbolic Address

A symbolic address has the form

C_5	$C_4C_3C_2C_1$

where the C's may be any character acceptable to the special character device, subject to the following restrictions:

a) C_5 must not be blank

b) If C_5 is numeric it cannot be followed by 4 numeric characters

c) An address meant to be symbolic must not have a regional form, i.e., an alphabetic character followed by four numeric characters.

Typical symbolic addresses are:

S	TART
S	KIP
X	
X	Y

E	63
P	4S25
L	0C3Q
P	1

Symbolic addresses are assigned optimal drum equivalents when first encountered in the assembly. When a symbolic address recurs, it is given the equivalent initially assigned to it.

It is possible to preassign a value to a symbol by use of a symbolic assignment card (see Symbolic Assignment Card, page 6).

Absolute Address

Absolute addresses have the form | |NNNN|

where NNNN is a valid four digit machine address (0000-1999). The symbolizer part of an absolute address must be blank.

Drum locations corresponding to absolute addresses are not made unavailable during assembly. Thus locations containing instructions or data specified by absolute addresses must be block reserved at the beginning of assembly.

Blank Address

Whenever either the D or I address of an instruction sends control to the "next" instruction written on the coding form, this address may be left blank. The location of the "next" instruction may be left blank or a symbol inserted, provided it has not been assigned previously, and the location will be given the same address as the D or I address left blank in the previous instruction. The program MUST be assembled without altering the order of these instructions. No instructions or data should be interspersed between two instructions related through blank addresses. These rules also apply to the location of data referred to by the data address of the preceding instruction.

The D and I address of an instruction should not both be blank; if they are the location of the next instruction will be the same as the previous I address. The location of the first instruction should not be blank.

An address should not be left blank in an instruction card unless the above meaning is specifically intended.

Operation Codes

Operation codes may be written in either three character symbolic or two digit numeric form. Symbolic operation codes to be used with SOAP are given in the appendix, page 20. Note that in general they differ from the symbolic operation codes given in the 650 manual.

The format for numeric operation codes is:

| |XX|

where (XX) is a valid 650 operation code, e.g., 65, and the symbolizer part of the operation field is blank.

If the D or I address of an instruction is to be assigned an optimal value by the assembly program, a symbolic operation code, e.g., RAL, must be used. If a numeric operation code is used, these addresses will be optimized according to the rules,

$$D = L + 5$$
$$I = D + 5$$

Numeric operation codes should be used only for constants.

Numeric Data

Numeric data is written using the absolute part of the OP, D, and I fields, e.g.,

Col 40	41	42 43—46	47 48 49	50 51—54	55	56 57—60	61	62	63—72
x	TYPE	Location	Oper.	Data Address	TAG	Inst. Address	TAG	SIGN	Remarks
x		PI	31	4159		2654			
x		ONE	10	0000		0051			
x		E	02	7182		8183			
x		1850	33	3333		3350		-	

The symbolizer part of these fields must be blank. Leading or trailing zeros must be punched in the card.

Tags

Whenever a D or I address is to be tagged, an A,B, or C should be inserted into the corresponding TAG position. If an absolute address is used in the D or I position it should be a valid drum address (0000-1999). If a SYMBOLIC address is used in the D or I position it must be remembered that only one (1) location is assigned to a symbol. If additional addresses are to be used, each additional address must have been reserved previously and a symbol preassigned. In this case it may be more convenient to use a preassigned REGIONAL address.

Alphabetic Data

Alphabetic data cards allow the programmer to enter alphabetic information into the 650 on load cards, up to five characters per card. Any characters acceptable to the special character device are permissible, and are written in the D field. During assembly, alphabetic data is converted to its numerical equivalent and the latter is punched as a ten digit number in exactly the same manner as numeric data. In the first example of figure 3

Col	40	41	42 43 — 46	47 48 49	50 51 — 54	55	56 57 — 60	61	62	63 — 72
	x	TYPE	Location	Oper.	Data Address	TAG	Inst. Address	TAG	SIGN	Remarks
	x		C 16		I BMNY					
	x		1306		CPC2					

Figure 3.

the number 69,61,74,75,88 will be loaded into symbolic location (C16).

Similarly, the 2nd example will load 00,63,77,63,92 into location 1306. The OP field of an alphabetic data card must be blank.

Signs

An 11 punch in column 62 signifies a negative instruction or negative data. This may be indicated by a minus sign, -, in the sign column on the Coding Form. A blank in column 62, or in fact any character not containing an 11 punch, will be interpreted as positive.

Symbolic Assignment Card: Type 4

Any symbol may be assigned an absolute equivalent prior to assembly by use of a symbolic assignment card.

As seen in Figure 4, a symbolic assignment card contains a symbol in the Location field and its desired four digit equivalent in the absolute part of the D field. The symbolizer position of the D field (* in Figure 4) should be punched "U" if the symbol represents a drum location to be made unavailable. Otherwise, this position must be blank.

Col	40	41	42 43 46	47 48 49	50 51 54	55	56 57 60	61	62	63 72
	x	TYPE	Location	Oper.	Data Address	TAG	Inst. Address	TAG	SIGN	Remarks
	x	4	S YMBOL		* ←equiv→					Symbolic Assig.

Figure 4.

Example 1

We wish to compute $$S = \sum_{i=1}^{m} x_i y_i$$

where $$x_i = a_i x_{i-1} + C \ , \ x_o = 0$$

$$y_i = b_i y_{i-1} + D \ , \ y_o = 0$$

$$m = 10$$

and the constants a_i, b_i, C and D are prescribed by the programmer.

We are given the following information:

m = 0000000010

a_i, b_i, C, D = .xxxxxxxxyy

where yy is the exponent

Figure 5 shows the flow chart for the proposed solution while the program is given in Figure 6. The assembled program is given in Figure 6A. The first four columns on the right of the listing show the assembled absolute program. The last column on the right is an assembled card number. The left side of the listing r eproduces all information on the coding form.

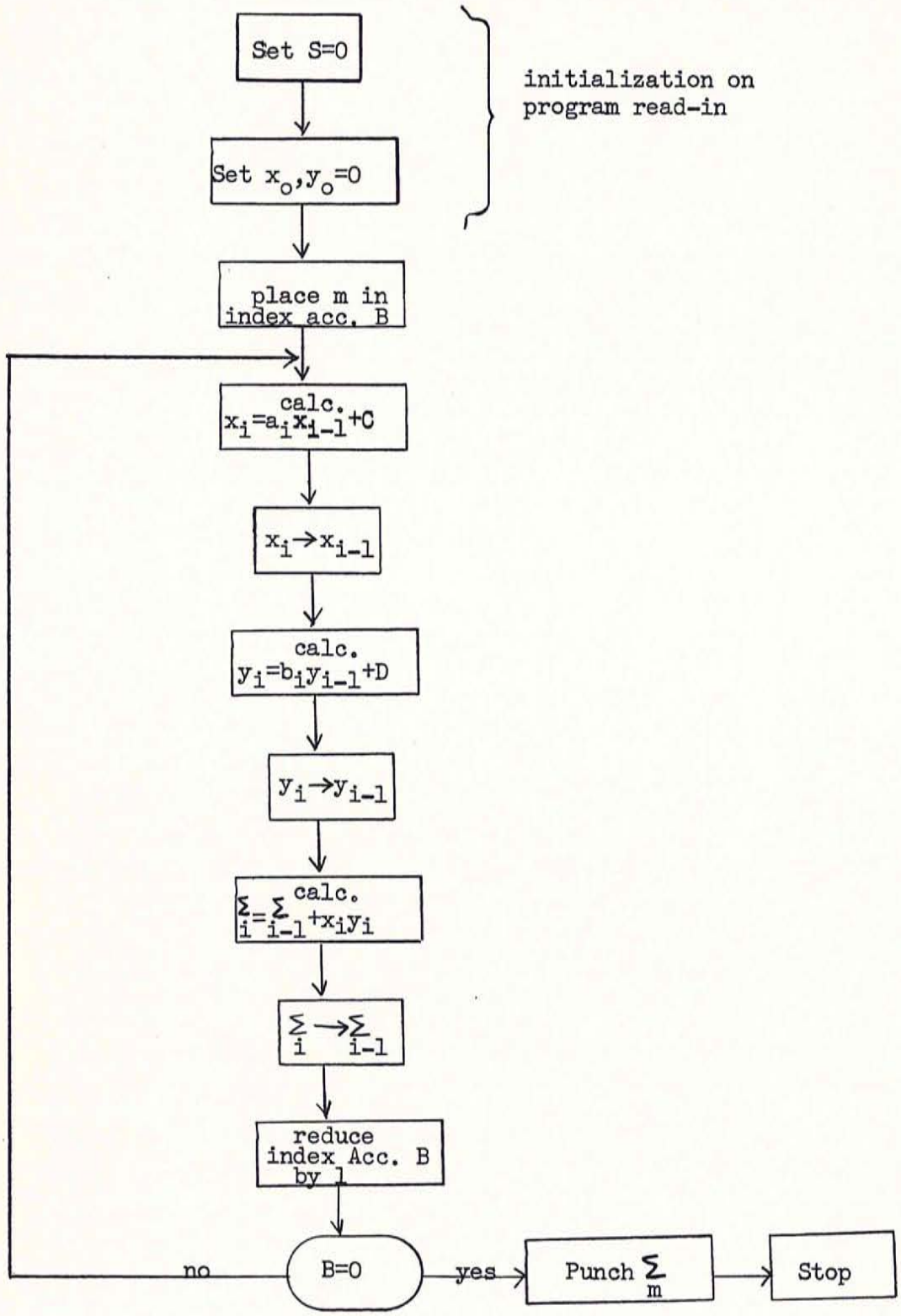

Set S=0
Set $x_o, y_o=0$
initialization on program read-in
place m in index acc. B
calc. $x_i=a_i x_{i-1}+C$
$x_i \rightarrow x_{i-1}$
calc. $y_i=b_i y_{i-1}+D$
$y_i \rightarrow y_{i-1}$
calc. $\Sigma_i=\Sigma_{i-1}+x_i y_i$
$\Sigma_i \rightarrow \Sigma_{i-1}$
reduce index Acc. B by 1
B=0
no
yes
Punch Σ_m
Stop

Figure 5

Col 40	41	42 43—46	47 48 49	50 51—54	55	56 57—60	61	62	63—72
x	TYPE	Location	Oper.	Data Address	TAG	Instr. Address	TAG	SIGN	Remarks
x	1			1900		0100			LD PCH RT
x	3	A		0000		0010			RESERVE A
x	3	B		0010		0010			RESERVE B
x	4	START		U 0050					FIRST INST.
x		START	LDD	M					
x			RSB	8001					Set Index
x		AAAA	RAU	X i					
x			FLM	A 0010	B				
x			FAD	C					
x			STU	X i					STORE Xi
x			RAU	Y i					
x			F LM	B 0010	B				
x			FAD	D					
x			STU	Y i					STORE Y i
x			F LM	X i					XIYI
x			FAD	S ANS					
x			STU	S ANS					STORE SUM
x			ADB	0001					reduce IND
x			NZB	A AAAA					TEST
x			STU	1977					
x			LDD			B BBBB			Set Control
x			00	8888		8888			word
x		B BBBB	STD	1986					
x			LDD	Z ERO					Clear 2nd
x		C CCCC	STD	1978					word
x			PCH	1977		9999			PUNCH
x									
x		X i	00	0000		0000			X Zero
x		Y i	00	0000		0000			Y Zero
x		S ANS	00	0000		0000			
x		Z ERO	00	0000		0000			

Figure 6

Program for example 1

1		1900	0100	LD PCH RT	9000				1
3	A	0000	0010	RESERVE A	9000				2
3	B	0010	0010	RESERVE B	9000				3
4	START	U0050		FIRST INST	9000				4

	Loc	OP	D	I	REMARKS	Loc	OP	D	I	
	START	LDD	M			0050	69	0053	0056	5
		RSB	8001		SET INDEX	0056	83	8001	0062	6
	AAAAA	RAU	XI			0062	60	0065	0069	7
		FLM	A0010 B			0069	39	4010	0020	8
		FAD	C			0020	32	0023	0039	9
		STU	XI		STORE XI	0039	21	0065	0068	10
		RAU	YI			0068	60	0021	0025	11
		FLM	B0010 B			0025	39	4020	0030	12
		FAD	D			0030	32	0033	0049	13
		STU	YI		STORE YI	0049	21	0021	0024	14
		FLM	XI		XIYI	0024	39	0065	0075	15
		FAD	S ANS			0075	32	0028	0044	16
		STU	S ANS		STORE SUM	0044	21	0028	0031	17
		ADB	0001		REDUCE IND	0031	52	0001	0036	18
		NZB	AAAAA		TEST	0036	42	0062	0040	19
		STU	1977			0040	21	1977	0080	20
		LDD		BBBBB	SET CONTRL	0080	69	0083	0086	21
		00	8888	8888	WORD	0083	00	8888	8888	22
	BBBBB	STD	1986			0086	24	1986	0089	23
		LDD	ZERO		CLEAR 2nd	0089	69	0041	0094	24
	CCCCC	STD	1978		WORD	0094	24	1978	0081	25
		PCH	1977	9999	PUNCH	0081	71	1977	9999	26
	XI	00	0000	0000	X ZERO	0065	00	0000	0000	27
	YI	00	0000	0000	Y ZERO	0021	00	0000	0000	28
	S ANS	00	0000	0000	SUM	0028	00	0000	0000	29
	ZERO	00	0000	0000		0041	00	0000	0000	30

Figure 6 A

USE OF LIBRARY PROGRAMS

Within the framework of SOAP, library programs are in either symbolic or relocatable form. Little need be said concerning symbolic library programs. They are written in exactly the same form as the main symbolic program and should be treated accordingly. It is important to note that such programs have no "guaranteed" optimization built into them.

Whenever tight optimization is required, one should use library programs written in relocatable form. Such programs are coded in absolute and are wholly contained in a block beginning at (0000) and extending to some specified last word address (LWA). They may be translated an amount specified by the programmer. This is accomplished by use of a Library Translation Card.

Library Translation Card: Type 7

This card contains the last word address (LWA) of the block to be translated and the amount of translation (Δ) punched in the absolute part of the D and I fields respectively, as shown in Figure 7. If Δ is negative, the word must be written negatively (col. 62).

Col	40	41	42 43—46	47 48 49	50 51——54	55	56 57—— 60	61	62	63 ——————72
	x	TYPE	Location	Oper.	Data Address	TAG	Instr. Address	TAG	SIGN	Remarks
	x	7			←LWA→		←Δ→			

Figure 7: Library Translation Card Format

A library translation card must immediately precede a relocatable library program and all such programs must precede the main program. Locations used by relocatable library programs are automatically made unavailable to the main program.

Example

The following are typical specifications for a relocatable library subroutine for calculating the square root of a floating point number.

Library Program 308 SR: $\sqrt{A}$

Conditions upon entry:

Contents of 8001: EXIT INSTRUCTION
8002: Clear
8003: A in normalized form

Conditions upon exit:

Contents of 8001: Irrelevant
8002: Clear
8003: $\sqrt{A}$ in normalized form

Transfer to 308 SR (0064) exit stored in 308xx(0026)
This subroutine occupies location 0000 to 0079
Index register A is used but is restored to orginal status before exit.
This subroutine should be translated by an even amount to preserve optimization.

Since the lowest location used by a relocatable library program is always (0000), and in this example LWA = 0079, it follows that this routine requires an 80 word block.

If the routine is to occupy the block (1210, 1289), the following library translation card should precede the square root deck:

Col	40	41	42 43—46	47——49	50 51—— 54	55	56 57——60	61	62	63——————72
	x	TYPE	Location	Oper.	Data Address	TAG	Instr. Address	TAG	SIGN	Remarks
	x	7			0079		1210			

A segment of the main program using this routine might be as follows:

Col	40	41	42 43 46	47——49	50 51——54	55	56 57——60	61	62	63——————72
	x	TYPE	Location	Oper.	Data Address	TAG	Instr. Address	TAG	SIGN	Remarks
	x			R AU	X					CALC SQ
	x			F AD	Y					ROOT of
	x			F SB	Z					X PLUS Y
	x			L DD			308SR			MINUS Z
	x			S TU	E 2					STORE ANS.

When coded as above, the exit instruction [STU(E2) ( )] will be optimized with respect to its drum location although it is actually executed from erasable location (308XX) in which it is placed by the subroutine. Optimization with respect to this latter location can be effected in the following manner which utilizes a pseudo-instruction located in (308XX):

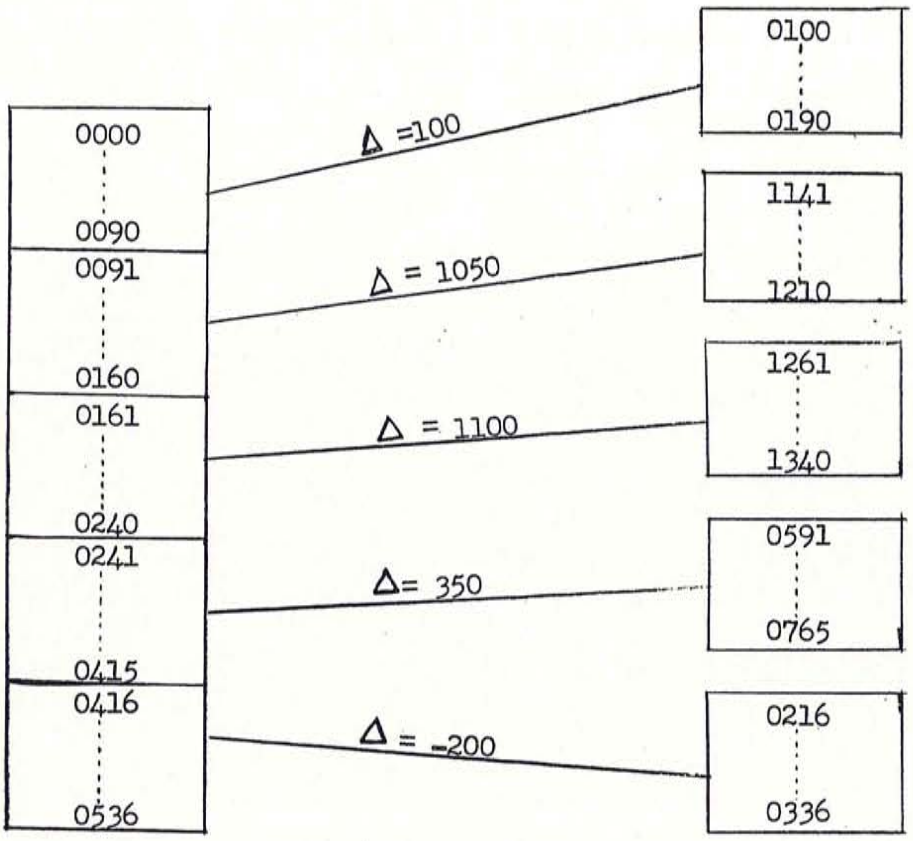
Col	40	41	42 43—46	47——49	50 51——54	55	56 57——60	61	62	63——————72
	x	TYPE	Location	Oper.	Data Address	TAG	Instr. Address	TAG	SIGN	Remarks
	x			R AU	X					CALC SQ
	x			F AD	Y					ROOT of
	x			F SB	Z					X PLUS Y
	x			L DD	E XIT		308SR			MINUS Z
	x		308XX	S TU	E 2		0000			
	x		E XIT	S TU	E 2					STORE ANS.

It is generally unnecessary to remove the pseudo-instruction [STU(E2) (0000)] from the assembly output deck. It will merely result in the loading of a spurious instruction into an erasable location.

A relocatable library program need not be entirely translated by a single amount Δ. It is possible, within certain restrictions which may be imposed by the library program, to split such programs into as many as five sub-blocks and translate each sub-block by an independent amount. A library translation card is required for each sub-block, LWA being the last location within the sub-block. These library translation cards are placed in front of the library program deck in ascending sequence on LWA.

Example:

Due to space limitations, a relocatable library program with LWA = 0536 is to be split into five parts and translated as follows:



The following library translation cards must precede the library deck when assembling:

Col	40	41	42 43 46	47 48 49	50 51 54	55	56 57 60	61	62	63 72
	x	TYPE	Location	Oper.	Data Address	TAG	Instr. Address	TAG	SIGN	Remarks
	x	7			0090		0100			
	x	7			0160		1050			
	x	7			0240		1100			
	x	7			0415		0350			
	x	7			0536		0200		-	

Note that all sub-blocks have been translated parallel to the drum axis in order to maintain optimization.

Translation may include a rotation about the drum axis provided all sub-blocks in a given program are rotated by the same amount, if optimization is to be preserved.

Relocatable Library Programs

As previously mentioned, relocatable library programs are coded in absolute and are wholly contained in a block extending from location (0000) to location(LWA). In the interest of tight optimization, all locations in this block need not be used. Unused locations are automatically available for the main program.

Two special types of cards are used in writing relocatable library programs:

Relocatable Library Card: Type 8

This card contains an absolute instruction or data located absolutely. The operation code of instructions MUST be numeric. The symbolizer part of the L,D, or I address must be blank if the corresponding address is to be translated; it should be punched "F" if the corresponding address is fixed. All addresses and only addresses in the L field are made unavailable to the main program. If L is relocatable then location (L + Δ) is made unavailable. Instructions located in 800X may be written on the coding form as a programming aid but must not appear in the final library deck.

All Tagged addresses should be written in 650 language(address plus 2000, 4000 or 6000).

Library Symbol Card: Type 9

Library symbol cards provide for the symbolic linkage of the main program with the library programs. These cards have somewhat the same format as symbolic assignment cards, i.e., a symbol in the L field and its untranslated drum equivalent in the absolute part of the D field. The symbolizer part of the D field should be blank. During assembly, the drum equivalent is translated in exactly the same manner as the library program and the symbolic address is stored in the symbol table along with the translated equivalent.

This card may either precede or follow the relocatable library program but must be inserted before another library program is translated.

The format for these two types of cards is shown in Figure 8.

Col	40	41	42 43 —— 46	47 48 49	50 51 —— 54	55	56 57 —— 60	61	62	63 —————— 72
	x	TYPE	Location	Oper.	Data Address	TAG	Instr. Address	TAG	SIGN	Remarks
	8		0 XXXX	XX	0 XXXX		0 XXXX			LIB TRANS.
	9		SYMBOL		←EQUIV→					LIB SYMB.

XX = digit, 0 = "F" (fixed) or blank (relocatable)

Figure 8: Card format for Relocatable Library Programs

To illustrate a relocatable library program, the short load routine "00SLR" has been written in SOAP form as shown in Figure 9. Note that a library symbol card is used in this example to symbolically specify the entry point to the subroutine, and library translation cards to illustrate how they are used.

Some library programs are not arbitrarily relocatable. For example if a table look-up is involved, or read and punch bands used within the routine, the entire routine must be translated by a multiple of 50 in order to preserve optimization and for proper operation of the program.

The example of the short load routine need not follow these rules specifically since optimization is of no value. However, since the read band must be translated by a multiple of 50, location 1951→1960 are translated by amount (-1950). All other locations are translated by amount (-1976). The final translated program will occupy locations 0001 → 0023.

650 S.O.A.P. Program Sheet

Problem Short Load Routine Written By _____

Col	40	41	42 43——46	47 48 49	50 51——54	55	56 57——60	61	62	63——72
	x	TYPE	Location	Oper.	Data Address	TAG	Instr. Address	TAG	SIGN	Remarks
	x	7			1960		1950		-	
	x	7			1999		1976		-	
	x	8	1996	70	1993		1993			READ CARD
	x	8	1993	67	1951		1994			
	x	8	1994	80	F 8002		1995			LOCATION
	x	8	1995	35	F 0006		1988			
	x	8	1988	44	1991		F 8001			
	x	8	1991	82	F 8003		1999			NOT TRANS
	x	8	1999	50	F 4000		1992			
	x	8	1992	51	F 0001		1997			
	x	8	1997	69	5951		1990			STORE
	x	8	1990	53	F 0001		1998			CONTENTS
	x	8	1998	24	F 2000		1989			ON DRUM
	x	8	1989	42	1992		1996			
	x	9	0 OSLR		1996					

Figure 9: Library Program

Assembly Capacity and Speed

During assembly all symbolic addresses and their equivalents are stored in a symbol table which can accomodate 600 symbols. As soon as the symbol table is filled the machine will stop with 1357 displayed in the data address of the instruction. All subsequent output cards will have a Y punched in Col. 40. Generally Y - 40 cards will be only partially assembled and should be fed into the 533 for a second pass, or more if required, until the last output card does not have a Y - 40. Intermediate Y - 40 cards should then be discarded.

Thus a program containing not more than 600 symbols can be assembled in one pass. There is no limit to the number of symbols provided a sufficient number of passes are made to complete the assembly.

Since the average program will require one symbolic address for every three to five instructions, programs containing 1,800 to 3000 instructions can generally be assembled in a single pass.

Assembly progresses at the rate of 75 to 100 cards per minute depending largely on the number of symbolic addresses present in the program and the degree of program packing requested by the programmer. It is preferable for the program to be assembled in the lower part of the drum if possible, as this will result in somewhat higher assembly speed. The speed will decrease toward the end of lengthy assemblies if the symbol table becomes densely packed.

Post Assembly Availability

To facilitate additions or corrections to an assembled program, the programmer can request an availability punchout upon termination of assembly. This will result in the punching of fifty "check-off" cards.

Check-off Card:

This card contains forty columns, (11 - 50), each punched one or zero, indicating respectively the availability or unavailability of the forty words comprising a specified dynamic drum level, e. g. level 00 = 0000,0050,----------, 1900, 1950. The level is punched in cols 57 - 60.

When listed, these fifty check off cards reveal at a glance all locations used by the program and those remaining for additions or corrections.

Since SOAP will load check-off cards and restore the availability-unavailability status which existed following a prior assembly, additional assembly at some future time is possible. Note however that symbolic equivalents cannot be restored in a similar manner.

It is possible to do a reassembly using output cards of a previous assembly as input, provided they are gang punched with X - 40.

Order of Assembly Deck

The assembly deck should have the following order:

1) SOAP
2) Deck to be assembled

Due to the one pass nature of the assembly, priority for the choice of optimal locations decreases as the assembly progresses. Thus, frequently executed portions of the main program should be placed toward the beginning of the assembly deck. It is also desirable to initialize a loop following the loop rather than before it.

In order to avoid reservation errors, check-off cards should be loaded immediately following the SOAP deck. The program should be started at location 0965 rather than 1000 if previous check off cards are used.

Machine Operator's Guide

1) 533 Read-Punch Unit
   a) Ready read feed with assembly deck
   b) Ready punch feed with blanks

2) 650 Console
   a) Set half-cycle switch to RUN
   b) Set display switch to PROGRAM REGISTER
   c) Set overflow switch to STOP
   d) Set error switch to STOP
   e) Set programmed switch to STOP

If SOAP is being loaded:
3A) Set [70, 1951, 1951] in storage-entry switches

If SOAP is not being loaded:
3B) Set [00,0000,1000] in storage-entry switches

4) Press computer-reset key

5) Press program-start key

6) If before completion of first pass, there was an error stop with 1357 displayed in data address of instruction, a second pass will be required.

Therefore proceed as follows:

a) Run out and remove completely assembled cards from card punch.
b) Press punch start key.
c) Press the program start key.
d) When read hopper empties, press end-of-file key.
e) On completion of first pass, run out partially assembled card from punch feed and insert in read feed.
f) Press Punch start key.
g) Set [00,0000,1500] in storage entry switches
h) Perform steps (4), (5), and (6).

If a second pass is not required proceed to step 8

7) After completion of all additional passes, discard all intermediate, (partially assembled) cards (Y - 40) and combine all completely assembled cards.

8) Print all final assembly cards (No Y - 40) on 402 with Set-up switch #1 on.

9) Sort completed cards on Col 7 and remove all cards which fall in 8 or 9 pocket. These should be discarded as they cannot be loaded into the 650.

10) The remaining cards are ready to be loaded into the 650 as a program deck.

Programmed Stops

Data Address	Reason and Procedure
1357	2nd pass required - if start key depressed assembly will continue, (see machine operators guide for complete instruction).
9999	Drum is full - Terminate assembly.
8888	Availability punch out completed

ERROR MARKS

ERROR MARKS	Reason
0001	Illegal operation code used. Operation of 00 was inserted on card and D and I address were optimized using the following rules., D = L + 5, I = D + 5
0002	A location of a library card was greater than LWA given on translation card. 0000 was inserted for location.
0003	The translated location of a library card was a negative location. 0000 was inserted for location.
0005	The symbol on a symbolic assignment card had already been defined. The original definition was retained.
0008	The symbol table was full for a symbolic assignment card. Symbol was not added to table.

650 OPERATION CODES

OPERATION	NUMERIC	SYMBOLIC
NO OPERATION	00	NOP
STOP	01	HLT
FLOATING ADD SUPPRESS NORMALIZATION	02	FAS
ADD TO UPPER	10	AUP
SUBT. FROM UPPER	11	SUP
DIVIDE	14	DIV
ADD TO LOWER	15	ALO
SUBT. FROM LOWER	16	SLO
ADD. ABS. TO LOWER	17	AAB
SUBT. ABS. FROM LOWER	18	SAB
MULTIPLY	19	MPY
STORE LOWER	20	STL
STORE UPPER	21	STU
STORE DATA ADDRESS	22	SDA
STORE INSTR. ADDRESS	23	SIA
STORE DISTRIBUTOR	24	STD
SHIFT RIGHT	30	SRT
SHIFT AND ROUND	31	SRD
FLOATING ADD	32	FAD
FLOATING SUBTRACT	33	FSB
FLOATING DIVIDE	34	FDV
SHIFT LEFT	35	SLT
SHIFT AND COUNT	36	SCT
FLOATING ADD ABS.	37	FAA
FLOATING SUBT. ABS.	38	FSA
FLOATING MULTIPLY	39	FLM
BRANCH NON-ZERO A	40	NZA
BRANCH MINUS A	41	MIA
BRANCH NON ZERO B	42	NZB
BRANCH MINUS B	43	MIB
BRANCH NON ZERO UPPER	44	NZU
BRANCH NON ZERO ACC.	45	BNZ
BRANCH ON MINUS	46	BMI
BRANCH ON OVERFLOW	47	BOV
BRANCH NON ZERO C	48	NZC
BRANCH MINUS C	49	MIC
ADD A	50	ADA
SUBTRACT A	51	SBA
ADD B	52	ADB
SUBTRACT B	53	SBB
ADD C	58	ADC
SUBTRACT C	59	SBC
RESET ADD UPPER	60	RAU
RESET SUBT. UPPER	61	RSU
DIVIDE RESET UPPER	64	DVR
RESET ADD LOWER	65	RAL
RESET SUBT. LOWER	66	RSL
RESET ADD ABS. TO LOWER	67	REA
RESET SUBT. ABS. FROM LOWER	68	RES
LOAD DISTRIBUTOR	69	LDD
READ	70	RDS

OPERATION	NU-MERIC	SYMBOLIC
PUNCH	71	PCH
RESET ADD A	80	RAA
RESET SUBT. A	81	RSA
RESET ADD B	82	RAB
RESET SUBT. B	83	RSB
TABLE LOOK UP	84	TLU
RESET ADD C	88	RAC
RESET SUBT. C	89	RSC
BRANCH ON 8 IN DIST. 1	91	BD1
BRANCH ON 8 IN DIST. 2	92	BD2
.		
.		
.		
.		
BRANCH ON 8 DIST. 10	90	BDØ *

* This is a zero

Optimizing 800X Instructions

In order that a program might be continuous, all instructions executed from an 800X address should appear on the coding form. Optimization of these instructions is not always possible since their D or I address is often a random location manufactured by the program. However when the range of these variables is known and reasonable, a certain amount of optimization can still be achieved if the maximum drum address is inserted in the variable location.

In the following example, the program has computed a shift in the D address part of the lower accumulator and the upper accumulator is to be shifted left by this amount. The coding shown in figure 10 causes the symbolic address (N5) of the mask (35,0000,N5) to be optimized as the I address of a SLT instruction with shift "9", located in the drum equivalent of 8002.

Col.	40	41	42 43—46	47 48 49	50 51——54	55	56 57—60	61	62	63——72
	x	TYPE	Location	Oper.	Data Address	TAG	Instr. Address	TAG	SIGN	Remarks
	x			A LO	M ASK		8002			
	x		8002	S LT	0009		N 5			
	x		N 5	~	~		~			
	x		M ASK	S LT	0000		N 5			

Figure 10: OPTIMIZING I-address of 8002 instr.

Note that if the pseudo-instruction (SLT,0009,N5) located in 8002 were not written, (N5) would be assigned a random drum location.

Thus a pseudo-instruction located in 800X may be used to force optimization of a manufactured instruction located in 800X provided it is written immediately following the instruction which sends control to 800X.

Optimizing Tagged Addresses

Tagged addresses, as 800X addresses, cannot always be optimized. However if a tagged data address refers to a block of data less than 50 words in length a certain amount of optimization is possible if the maximum drum location appears in the data address. This may be achieved by originally setting the index accumulator to a negative quantity.

In the example in figure 11, the program is to compute $\sum_{i=1}^{10} Q_i$. The Q's will be placed in drum location 1000 →1009, with Q_1 in location 1000, Q_2 in location 1001, etc.

Col.	40	41	42 43—46	47 48 49	50 51——54	55	56 57——60	61	62	63——72
	x	TYPE	Location	Oper.	Data Address	TAG	Instr. Address	TAG	SIGN	Remarks
	x	3	A		1ØØØ		ØØ1Ø			Reserve Q
	x			RSA	ØØ1Ø					Set INDEX
	x			RAU	Zero					Clear ACC.
	x		LOOP	AUP	AØØ1Ø	A				Compute
	x			ADA	ØØØ1					SUM
	x			NZA	LOOP		EXIT			

Figure 11: Tagged instructions

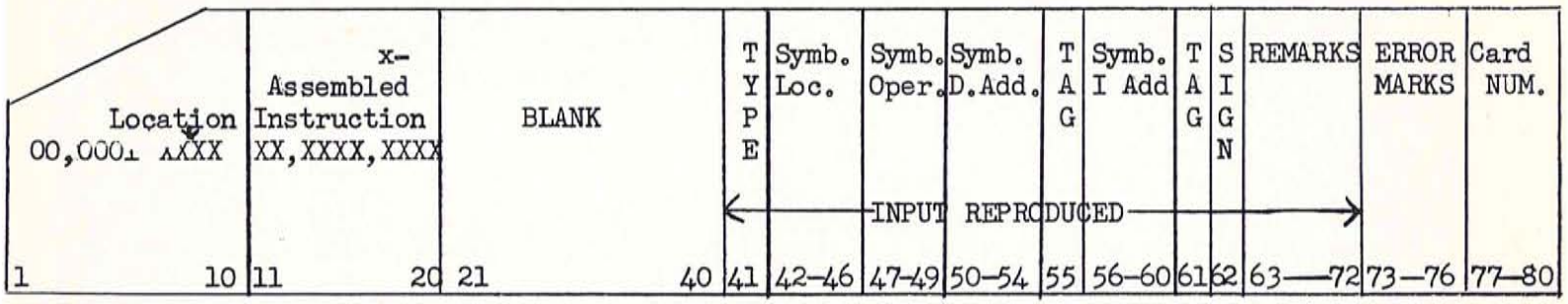
The index accumulator A is set to negative 10 and the Q's are successively added to the upper accumulator starting with Q_1 and finally adding in Q_{10}.

The I address of the instruction (LOOP) will be optimized as a AUP instruction with data address of 1010.

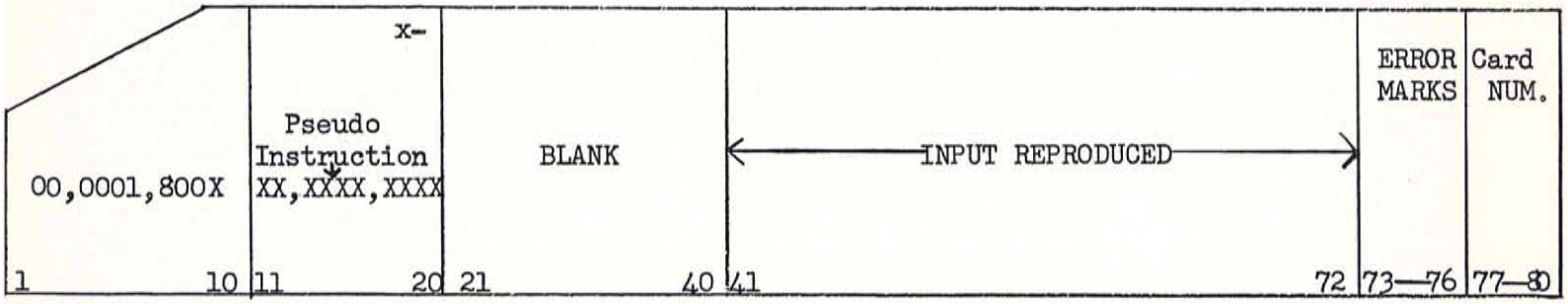
Note that if a plus quantity had been stored in index accumulator A this instruction would be optimized with data address of 1000 so that only the first instruction address would be optimized with respect to the data address.

Instructions with tagged instruction addresses are difficult to optimize, since these addresses must be preassigned. However, optimization will begin again with the new locations referred to by the tagged instructior address.

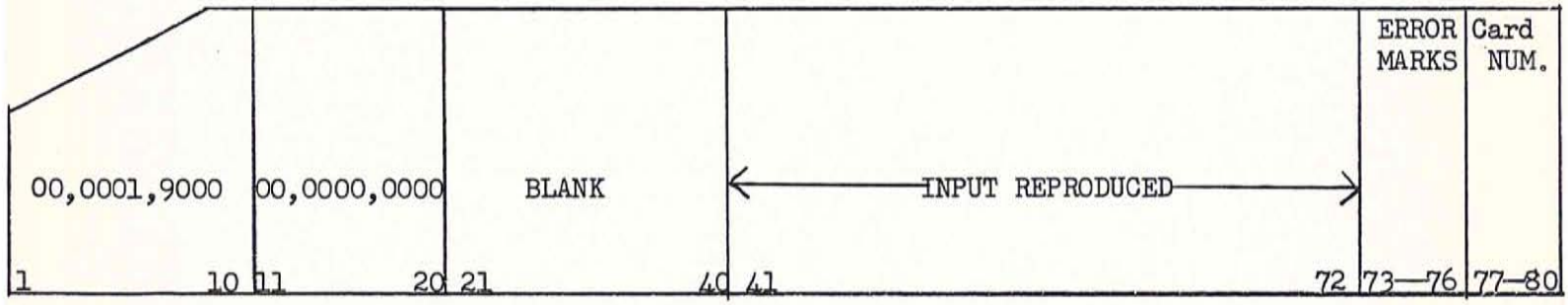
INSTRUCTION, DATA, and Type 8 cards



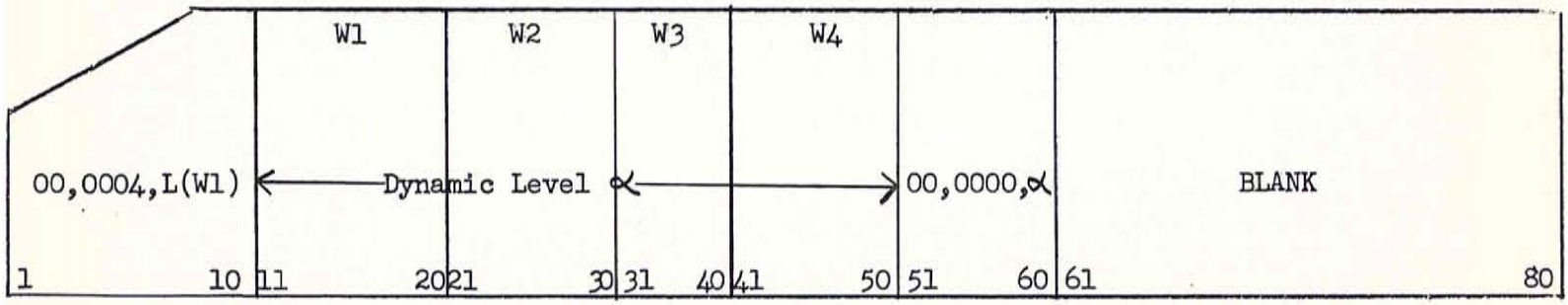
800X Pseudo Instructions



TYPE 1, 3, 4, 7, 9 Cards



Check - off Card $(0000 \leq \alpha \leq 0049)$



DESCRIPTION OF OUTPUT CARDS

Col.	40	41	42 43——46	47 48 49	50 51——54	55	56 57——60	61	62	63——72
	x	TYPE	Location	Oper.	Data Address	TAG	Instr. Address	TAG	SIGN	Remarks
	x	1			←FWA→		←△→			BLOCK RESERVATION
	x	3	*		←FWA→		←△→			REGIONAL SPECIFICATION
	x	4	SYMBOL		0←equiv→					SYMBOLIC ASSIGNMENT
	x	7			←LWA→		←△→		'S"	LIBRARY TRANSLATION
	x	8	⊡XXXX	XX	⊡XXXX		⊡XXXX			RELOCATABLE LIBRARY
	x	9	SYMBOL		←EQUIV→					LIBRARY SYMBOL

* = alphabetid character associated with region
0 = υ if symbol represents a drum location to be made unavailable; blank otherwise
'S" = sign of △, -(ιι) if negative; blank if positive
⊡ = F if fixed address or data; blank if relocatable
x = digit

Summary of Special type Cards

Standard Writing Procedure

In order to establish uniformity in writing symbolic coding and to eliminate errors in misread symbols, the following practice has been agreed upon:

I ALL LETTERS WILL BE WRITTEN IN CAPITAL LETTERS WITH THE EXCEPTION OF THOSE IN SECTION II

II THE FOLLOWING SET HAS BEEN DEVISED TO ELIMINATE ANY MISUNDERSTANDING BETWEEN KEYPUNCHERS AND PROGRAMMERS

Ø	ZERO
O	LETTER O
Q	LETTER Q, q,
i	LETTER I, i
l	NUMBER ONE
Z	LETTER Z, z

REMEMBER YOU HAVE TO DEBUG THE CODE SO WRITE LEGIBLY.