650 DATA PROCESSING SYSTEM BULLETIN

SOAP 2L, SOAP 2L TAPE, SOAP 4000, AND SOAP 42

This bulletin obsoletes the SOAP IIA bulletin, Form J24-4001 (as well as previous editions of the SOAP IIA bulletin).

SOAP 2L assembly program replaces SOAP IIA, and SOAP 2L Tape replaces Tape SOAP IIA. SOAP 2L and SOAP 2L Tape incorporate major improvements over the two programs they replace.

SOAP 2L is a multiple machine-pass assembly program. Programs containing any number of symbols can be assembled without the sectioning, prior to processing, that is required when programs using more than 400 symbols are assembled by SOAP II. SOAP 2L is used for program assembly employing a card input-output configuration IBM 650 Model 2 (2000-word drum) Data Processing System. SOAP 2L Tape is used for program assembly employing a 650 Model 2 tape system. Both programs can assemble programs written for any configuration of the 650 Model 2 system. They use the same basic processing methods differing mainly in speed, input-output procedures, and system requirements. SOAP 2L Tape is the faster of the two programs because it can use IAS for storing frequently-used constants and program loops and for temporary storage locations.

The minimum system requirement for running SOAP 2L is an IBM 650 Model 2 card input-output configuration equipped with the alphabetic special feature, special-character group I device, and a total of 12 co-selectors. In addition, SOAP 2L Tape requires at least two IBM 727 Magnetic Tape Units.

SOAP 4000 assembly program is used on an IBM 650 Model 4 (4000-word drum) system to assemble programs written for a 650 Model 4. SOAP 42 program is used on a 650 Model 2 (2000-word drum), to assemble programs written for a 650 Model 4. Most of the improvements and programming features of SOAP 2L and SOAP 2L Tape are incorporated in SOAP 4000 and SOAP 42. The minimum system requirement for these two assembly programs is the same as for SOAP 2L, except for the 650 drum capacity already given.

The information presented in this bulletin supplements that given in the IBM *Reference Manual SOAP II for the IBM 650 Data Processing System*, C24-4000. Only changes and additions to the SOAP II assembly program are presented. Other than these modifications, the information given in the manual still applies.

Requests for copies of condensed program decks and listings should specify which of the four versions (SOAP 2L, SOAP 2L Tape, SOAP 4000, or SOAP 42) is desired. Requests should be made in the usual manner, through the local IBM sales representative or sales office.

SOAP 2L and SOAP 2L Tape

Program Assembly

Basically, translation and assembly of symbolic program instructions by SOAP 2L are accomplished in the same manner as by SOAP II. However, SOAP 2L can be used to assemble long programs without the sectioning formerly required to insure that the number of symbols in any one section did not exceed a set maximum. When SOAP 2L is used, only the separation of the output into two parts (i.e., completely assembled and partially-assembled cards) is required. The partially-assembled cards are then used as input to the following machine pass. In the case of SOAP 2L Tape, the assembly process is completely automatic, and no card handling is required because only completely assembled cards are punched.

Specific information concerning the assembly of programs with SOAP 2L is included in the following paragraphs.

Input Cards

The specifications for input cards (i.e., symbolic instruction cards) for the *initial* pass of SOAP 2L are identical to those of SOAP II, except for one additional restriction: column 80 of the input cards must not contain a 12, 1, 2, 3, 4, 5, 6, or 7 punch. The presence of one or more of these punches in column 80 will cause improper assembly.

As will be seen, program cards assembled with SOAP 2L contain only a 9 punch in column 80. Therefore, symbolic programs can be reassembled with SOAP 2L.

Assembly

SOAP 2L and SOAP 2L Tape function the same as SOAP II up to the point at which the maximum number of symbols have been entered into the symbol table. At this point assembly is suspended and a partial-assembly phase is begun. During this phase no entries are made in the availability- or symbol-tables and no pseudo-operation code (except HED) is executed. Partial assembly consists of reading all symbols and checking these symbols against the symbol table. If a symbol is found in the table, its equivalent location is inserted into the output field (either of a card or of a tape record) reserved for the assembled instruction, and a digit is inserted into a control word as an indication that the symbol has been translated. At the beginning of the next pass the symbol table is cleared in preparation for the construction of a new table. The symbolic information contained in a partially-assembled card is reproduced into the appropriate output field.

During a machine pass, the indication that a given symbol has been previously translated prevents the program from entering that symbol into the symbol table and assigning it a new location thereby destroying program continuity. Retention of the symbolic information in its original form permits subsequent reassembly of a program.

SOAP 2L

When this version is used, the partial-assembly phase is begun following the assembly of the card containing the 228th symbol. During this phase, partially assembled instructions are punched out as described above, and the control digits are translated into a combination of punches (see *Output Cards*) that are punched into column 80 of the output cards.

Following each machine pass the output deck must be checked to determine whether the last output card contains a 9 punch in column 80 or whether it is an availability-table card. If either of these conditions is found, assembly is complete. Otherwise, all cards containing a 9 punch in column 80 must be removed from the output and held pending completion of assembly. (NOTE: All completely assembled cards will be together at the front of the output deck.) The remainder of the output deck is used as input for the next pass.

When all output cards of a pass are found to be assembled, the sections previously removed from output decks are placed together to form one program deck (see Figure 1). The final deck, when correctly placed together, will be numbered sequentially throughout the entire deck.

Example: Assume that the symbolic card shown in Figure 2 is read after assembly is suspended. Also assume that the symbols SOME and NEXT are found in the symbol table to correspond to locations 0542 and 1349, respectively. The corresponding output card is punched as shown in Figure 3.

SOAP 2L TAPE

During assembly of a program with this version, assembly is suspended following the card (or tape record) containing the 200th symbol. However, cards are punched out during the assembly phase only. At the beginning of the first partial-assembly phase, all remaining cards are read, partially assembled, and written on tape. Upon completion of this phase, the next assembly phase is begun automatically. Input to the program is taken from the tape previously written. Again during this pass assembled cards are punched, assembly is suspended following the processing of the record containing the 200th symbol, and incompletely assembled cards are written on tape. The process continues until the program is completely assembled. (See Figure 4 for schematic diagram.)

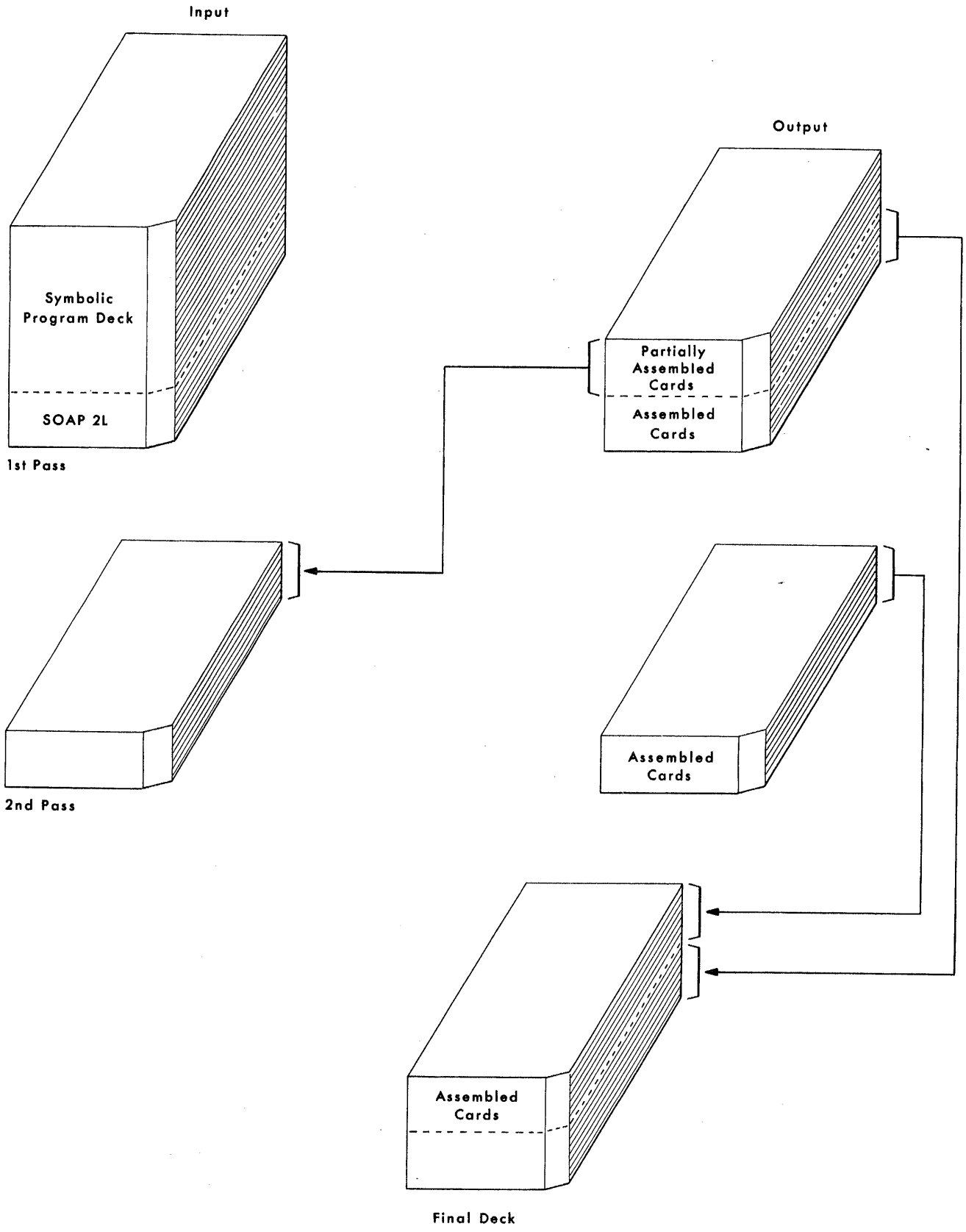


Figure 1. Program Assembly Using SOAP 2L (Source Deck Contains 229-456 Symbols)

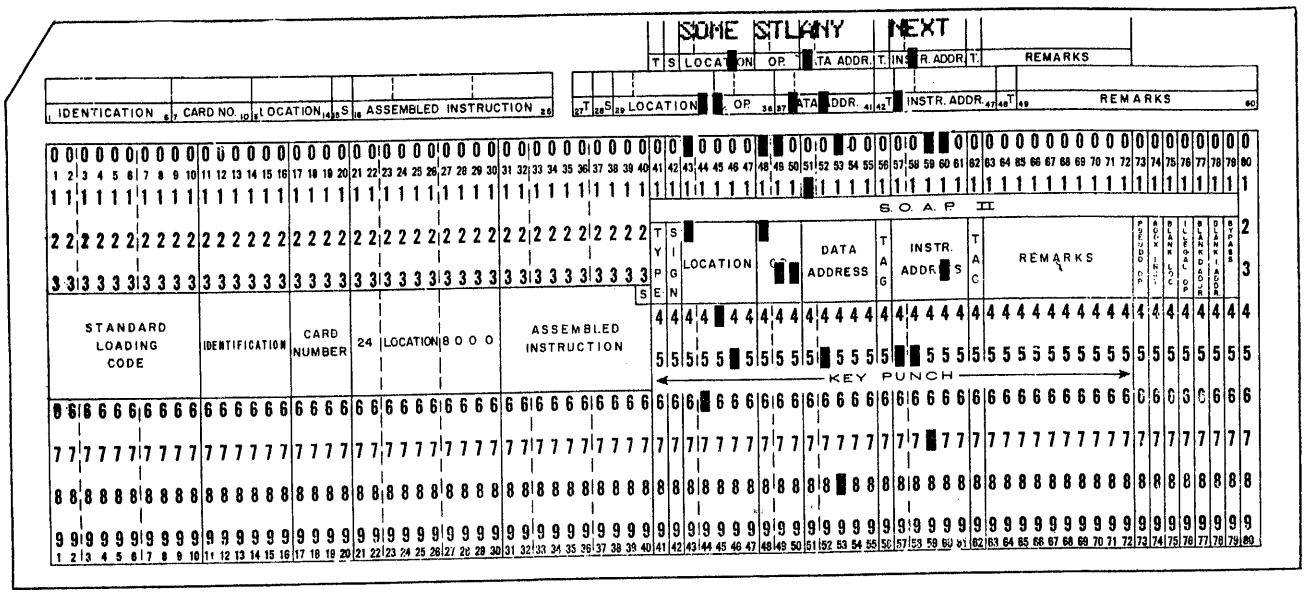


Figure 2. Symbolic Input Card

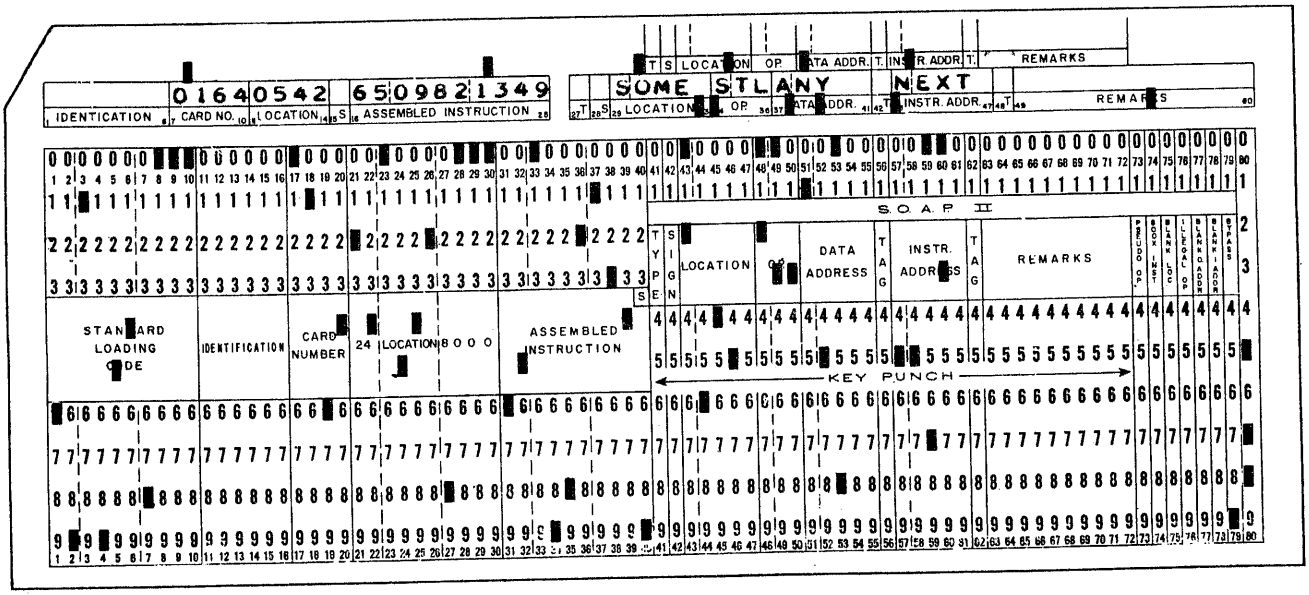


Figure 3. Partially-Assembled Output Card

In partially-assembled cards, the operation code punched into columns 31 and 32 has no significance; nor do any addresses punched into columns 23-26 and 33-40 which are not indicated, by the appropriate punch in column 80, as having been translated. The operation code will be identical with the operation code in the last completely-assembled card. Any addresses not translated will be the same as the last translated address of that field. For information pertaining to the punching in column 80, see **Output Cards.**

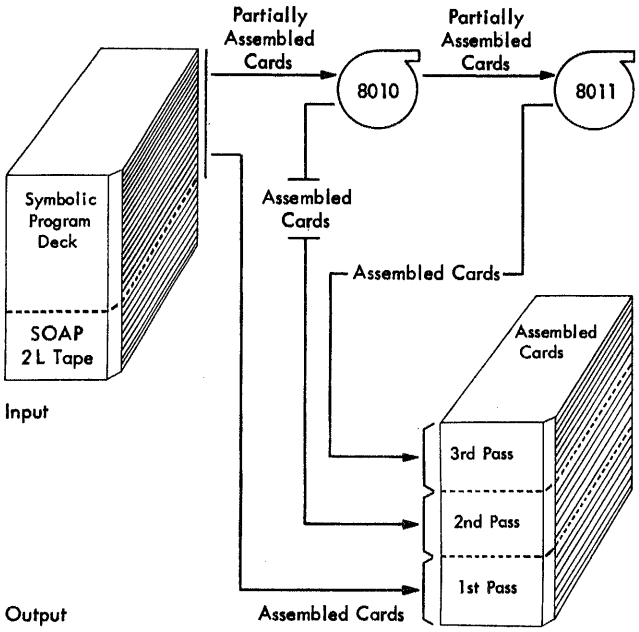


Figure 4. Program Assembly Using SOAP 2L Tape (Source Deck Contains 402-600 Symbols)

Symbol Table

With SOAP 2L the number of symbols contained in the symbol table at the start of partial-assembly phase will be 228, 229, or 230 (or 200, 201, or 202 with SOAP 2L Tape). This is because partial assembly is begun *after* assembly of the card containing the 228th (or 200th) symbol, and this card may have one or two new symbols in addition to the 228th (or 200th). Although no more than 230 (or 202) symbols are ever loaded into the table, an allowance is made for a few additional locations in the table. This allowance permits more rapid partial assembly than would be possible if the entire table were filled.

Output Cards

All output cards are punched with one or more identifying digits in column 80. These are:

Identification	*Meaning*
9	Completely-assembled cards (no other punches will appear in such cards).
8	Incompletely-assembled cards (may appear alone or in combination with 4, 5, 6, or 7).
7	I-address translated.
6	D-address translated.
5	Location translated.
4	First incompletely assembled card of the machine pass.

NOTE: Because only completely-assembled cards are punched by SOAP 2L Tape, only a 9-punch appears in column 80 of the output cards from that version.

Sequence Checking Assembled Program Deck

The cards (except the first five loading-routine cards) of the five-instructions-per-card program deck are sequence-numbered in columns 7-10. When the program deck is loaded, the cards are checked to see that they are in sequence and that none are missing. If either condition is detected, a programmed halt occurs with an instruction in the program register of the form 01 XXXX XXXX, where the I-address is the sequence number of the first card of the two being compared when the error is detected. If a sequence-error halt occurs, the restart procedure is to clear the read hopper, correct the sequence, and reload, starting at the beginning of the deck.

Miscellaneous: SOAP 2L

When used as program cards, incompletely-assembled cards will be passed and not loaded into any drum location. When listed on the IBM 407 Accounting Machine, such cards will cause BYPAS to be printed to the right of the remarks (except HED and comments cards). BYPAS will also be printed for EQU or SYN cards of the final deck, which were bypassed because of undefined symbolic or regional instruction addresses.

If an availability table is to be reloaded during a multiple-pass assembly to restore previous availability conditions, the table must be loaded before the 228th symbol is encountered.

The multifile-assembly procedure (i.e., the stacking of two or more symbolic programs with intervening BOP cards) described under *Multifile Assembly: BOP* in the *SOAP II Reference Manual* should not be used unless it is known that each program contains fewer than 228 distinct symbols.

Miscellaneous: SOAP 2L Tape

Programs assembled by SOAP 2L Tape must have, as the last program card, a card punched XYZ in columns 48-50 (symbolic op-code field). This card, however, will not appear in the output of the program.

If an availability table is to be reloaded during a multiple-pass assembly to restore previous availability conditions, the table must be loaded before the 200th symbol is encountered.

Programming Features

Additional Pseudo Instructions

In addition to the pseudo-operation codes described in the *SOAP II Reference Manual,* five other pseudo-instructions can be used. Their explanations and illustrations of their use follow.

DLA (DYNAMIC LEVEL AVAILABLE)

This pseudo operation makes available to the program the *equivalent dynamic locations* between, and including, the two locations specified in the D-address and the I-address of the DLA pseudo instruction. *Equivalent dynamic drum locations* are those occupying the same word position around the drum, regardless of the band (for example, 0014, 0064, 0114, 1914, 3964). The drum locations specified in the D- and I-address must be of the same dynamic level.

In Figure 5, the first DLA pseudo instruction makes available to the program the one drum location, 0048. The second DLA makes available to the program drum locations 0049, 0099, 0149, 0199, 0249, and 0299. Thus, if a large block of storage has been reserved for storing a table that is to be searched using a TLU operation, the last two storage locations in each band of the block can be made available to the program by using only two DLA pseudo instructions (see Figure 6).

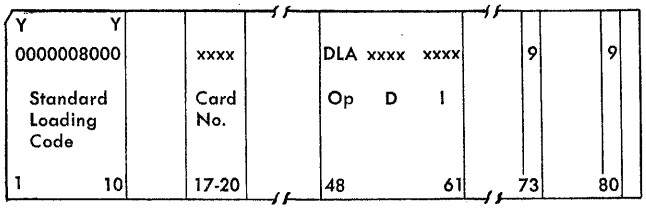
DLA cards are not only used as part of symbolic input to program assembly, but are produced as output

41	42	43	44- -47	48	49,50	51	52- -55		56	57- -60		61- -72	SOAP I COL.
41	42	43	44- -47	48	49,50	51	52- -55	56	57	58- -61	62	63- -72	SOAP II COL.
T P	S N	LOCATION		OPER. CODE		DATA ADDRESS		TAG		INSTR. ADDRESS	TAG	REMARKS	ACCU UPPER 8003
1		I	NITI	A	LI	Z	ATIO	N					
				-	- -	-	- - -						
				B	LR		0000			0499			
				D	LA		0048			0048			
				D	LA		0049			0299			
				-	- -	-	- - -						
				-	- -	-	- - -						

Figure 5. Single and Multiple Drum Locations Made Available

41	42	43	44- -47	48	49,50	51	52- -55		56	57- -60		61- -72	SOAP I COL.
41	42	43	44- -47	48	49,50	51	52- -55	56	57	58- -61	62	63- -72	SOAP II COL.
T P	S N	LOCATION		OPER. CODE		DATA ADDRESS		TAG		INSTR. ADDRESS	TAG	REMARKS	ACCU UPPER 8003
1		I	NITI	A	LI	Z	ATIO	N					
				-	- -	-	- - -						
				B	LR		1000			1799		TABLE	
				-	- -	-	- - -						
1		P	ROGR	A	M								
				-	- -	-	- - -						
				D	LA		1048			1798			
				D	LA		1049			1799			
				-	- -	-	- - -						

Figure 6. Last Two Drum Locations in Each Band of Table Area Made Available



Y represents a 12-punch.

Figure 7. DLA Output Card Format

from the assembly as well. Output DLA cards are produced as a result of the pseudo instruction, PAT, and they indicate the available storage locations by dynamic levels. The format of DLA output cards is shown in Figure 7.

Card number (columns 17-20) of all DLA output cards for a program is the same as the card number of the output PAT card that precedes them.

DLA output cards can be used to reinstate the availability table they represent. The procedure is to place an RDR card first, followed by the output DLA cards.

Because the availability table is punched out in the form of DLA cards, altering or updating the availability table is quite easy. Thus, an availability table can be altered to reflect a corrected storage status after *patching* a program, by appropriate removal, addition, or alteration to, DLA output cards. For example, if a user discovers after program assembly that one additional storage location is needed, he can examine his DLA output cards. He picks out a card that indicates one available storage location (for example, DLA 1610 1610), notes the address, and uses it to make the patch. He then discards the DLA card representing the location used. The remaining DLA output cards, if loaded as input following an RDR card, reinstate the corrected availability table for the assembled program.

One or more consecutive DLA cards immediately following a PAT card in the input deck will be bypassed. They will neither appear in the output as DLA cards nor affect the present availability table. Therefore it is not necessary to remove DLA output cards from a symbolic deck previously produced as output from an assembly, when the symbolic deck is to be used as input for reassembly.

At the completion of a PAT pseudo operation, the entire drum is unavailable to the assembly program, as though a reserve-drum pseudo instruction were given. Thus a measure of drum security is automatically effected, guarding against duplicate assignment of drum locations by the assembly program, or inadvertent overlay of information on the drum. To proceed with subsequent program assembly immediately after obtaining the availability table output, the user specifies the locations that are to be made available for further assembly. Such specification can be accomplished by using as input either (1) an appropriately-prepared BLA card, or (2) the appropriate DLA output cards obtained after a previous program assembly.

PST (PUNCH SYMBOL TABLE)

This pseudo instruction permits punching the contents of the symbol table together with the equivalent absolute addresses. The symbols and their equivalents are punched into the equivalence (EQU) card format.

Example: Assume the symbol table and equivalence table contain the following entries when a PST card is encountered.

Symbol Table	*Equivalence Table*
EDPM	00 0100 0462
TAX	
.	.
.	.
WAGES	
FICA	00 1807 0025

Cards would be punched as follows.

OP	*D-Address*	*I-Address*
PST	(blank)	(blank)
EQU	EDPM	0100
EQU	TAX	0462
.	.	.
.	.	.
.	.	.
EQU	WAGES	1807
EQU	FICA	0025

NOTE: 1. Symbols that are headed when entered into the symbol table will be headed when punched.

2. All EQU cards will be numbered 0000.
3. If a PST card is encountered during the partial assembly phase, no symbol table punchout will occur.

The SOAP 2L and SOAP 2L Tape programs also include a test of the 650 console sign switch. This is made at the beginning of the second and of each succeeding machine pass of a program assembly. If, when the test is made, the sign switch is set to minus (—), a punchout of the contents of the symbol table will occur. That is, all symbols used in the preceding pass will be punched into EQU cards.

Because the above described features are included in the program, and because the PST pseudo-op code is not executed during a partial assembly phase, a punchout of all symbols used in a program may be obtained by setting the sign switch to minus (—) and placing a PST card at the end of the input deck. This will cause a punchout of the contents of the table at the beginning of each machine pass (except the first) and upon completion of assembly.

If the only PST card included in the program is at the end of the input deck, no symbol will be punched more than once, unless redefined by an EQU or SYN card.

The EQU cards obtained from a symbol table punch-out have two principal uses.

A. The cards may be sorted on the symbols or equivalents and listed for reference. Such listings are useful for detecting incorrect absolute addresses and erroneous symbols not discovered during pre-assembly desk checking.
B. Some or all of the EQU cards, together with the availability table, can be reloaded to establish assembly conditions if it becomes necessary to incorporate changes or new symbolic sub-programs into an assembled program.

RDR (RESERVE DRUM)

This pseudo instruction rapidly reserves the entire drum. It is particularly useful in conjunction with assembling a program that requires less than half the storage capacity of the drum. When a symbolic program is ready to be assembled, an RDR card is placed ahead of the BLA card that designates the portion of the drum to be used for the assembly. Drum reservation (RDR) is completed substantially faster than block reservation (BLR). For example, an entire 2000-word drum can be reserved by performing an RDR pseudo instruction in three seconds, compared to the 180 seconds required for a BLR pseudo instruction. Figure 8 illustrates the use of the RDR pseudo operation in assembling a 250-instruction subroutine in drum locations 3700-3999.

The procedure formerly required (that is, not using the reserve drum pseudo operation RDR) was the slower method of reserving that part of the drum not required by the program. Figure 9 illustrates the block-reservation method formerly used.

SEQ (ASSIGN SEQUENTIALLY)

This pseudo operation causes the assembly program to assign sequential numbers to L-, D-, and/or I-address portions of instructions or data. For example, in *Part I, Figure 4, SOAP II Reference Manual,* the drum locations 0000-0009 are reserved for storing a rate table. The data composing the table must be placed on the drum. One way to do this is to assign the appropriate drum location for each rate. A better way is to use the pseudo operation SEQ as shown in Figure 10. The assembly program *automatically* assigns sequential L-address locations to the constants that make up the rate table. The first L-address assigned is that specified in the L-address part of the SEQ pseudo instruction. Assignment is terminated when the program encounters an SEQ pseudo instruction in which the L-, D-, and I-addresses are blank.

Sequential assignment of storage locations (drum and IAS) to L-, D-, and/or I-addresses of instructions and data by the assembly program is accomplished if:

1. An SEQ pseudo instruction with the desired starting location(s) in its L-, D-, and/or I-address, is used. In Figure 10, the pseudo instruction used is 0000 SEQ (blank) (blank), which will cause sequential number assignments in the L-addresses of the constants of the table starting with number 0000.

41	42	43 44- -47	48 49,50	51 52- -55		56 57- -60		61- -72	SOAP I COL.
41	42	43 44- -47	48 49,50	51 52- -55	56	57 58- -61	62	63- -72	SOAP II COL.
T P	S N	LOCATION	OPER. CODE	DATA ADDRESS	TAG	INSTR. ADDRESS	TAG	REMARKS	ACCU UPPER 8003
			R D R						
			B L A	3 7 0 0		3 9 9 9			

Figure 8. Reserving the Drum and Making Available a Block of Drum Locations

41	42	43 44- -47	48 49,50	51 52- -55		56 57- -60		61- -72	SOAP I COL.
41	42	43 44- -47	48 49,50	51 52- -55	56	57 58- -61	62	63- -72	SOAP II COL.
T P	S N	LOCATION	OPER. CODE	DATA ADDRESS	TAG	INSTR. ADDRESS	TAG	REMARKS	ACCU UPPER 8003
			B L R	0 0 0 0		3 6 9 9			

Figure 9. Reserving a Block of Drum Locations

41	42	43 44- -47	48 49,50	51 52- -55		56 57- -60		61- -72	SOAP I COL.
41	42	43 44- -47	48 49,50	51 52- -55	56	57 58- -61	62	63- -72	SOAP II COL.
T P	S N	LOCATION	OPER. CODE	DATA ADDRESS	TAG	INSTR. ADDRESS	TAG	REMARKS	ACCU UPPER 8003
			5 C D	0 0 0 2		0 0 1 1			
			B L R	1 9 5 1		1 9 6 0		READ AREA	
			R E G	P 0 0 2 7		0 0 3 0		PUNCH AREA	
			B L R	0 0 0 0		0 0 0 9		RATE TABLE	
		0 0 0 0	S E Q					STORE	
			0 0	0 0 0 0		0 1 2 6		TABLE	
			0 0	0 0 0 0		0 1 3 7			
			0 0	0 0 0 0		0 1 4 8			
			0 0	0 0 0 0		0 1 6 0			
			0 0	0 0 0 0		0 1 7 3			
			0 0	0 0 0 0		0 1 8 7			
			0 0	0 0 0 0		0 2 0 2			
			0 0	0 0 0 0		0 2 1 8			
			0 0	0 0 0 0		0 2 3 5			
			0 0	0 0 0 0		0 2 5 3			
			S E Q						
1									
		S T A R T	R C D	1 9 5 0				READ CARD	
			L D D	1 9 5 1				STORE	
			S T D	P 0 0 0 1				IDENT	
			- - -	- - - - -					
			P C H	P 0 0 0 1		S T A R T		PUNCH	
			X F R	S T A R T					

Assembled Table With Assigned L-Addresses

L	OP	D	I
	(----	TABLE	----)
0000	00	0000	0126
0001	00	0000	0137
0002	00	0000	0148
0003	00	0000	0160
0004	00	0000	0173
0005	00	0000	0187
0006	00	0000	0202
0007	00	0000	0218
0008	00	0000	0235
0009	00	0000	0253

Figure 10. Using SEQ to Assign L-Addresses of Table

2. The instruction or the data to which sequential numbers are to be assigned has either a *blank* L-, D-, and/or I-address or a *symbolic* L-, D-, and/or I-address *not previously defined.*

In connection with condition 2, if an instruction or a word of data contains either an absolute entry or a symbolic entry not previously defined in the L-, D-, and/or I-address, no sequential number(s) is assigned. Instead, the absolute entry or the previous definition appears in the L-, D-, and/or I-address of the assembled output for the instruction. Furthermore, the next sequential number assigned by the assembly program, after such an interruption as that just mentioned, is sequential as though the interruption had not occurred. Thus, sequential numbers are skipped in those instructions containing absolute entries (Figure 11, such as in the instruction NZA 9004 9006, where the IAS location 9004 is skipped) or symbolic entries previously defined.

Recommended practice is to reserve the block of drum locations that are to be assigned by the SEQ pseudo instruction. The user can do this at the beginning of the main program during initialization. In Figure 10, the drum locations were reserved by the pseudo instruction BLR 0000 0009.

Figure 11 illustrates the use of the SEQ pseudo instruction in assigning sequential drum L-addresses and sequential core I-addresses for a drum-stored core-executed closed subroutine.

If sequential assignment of table values is to be made, locations xx48 and xx49 should not be used (so that TLU operations function properly). Also, if assignment is to resume at locations xx50 or xx00, another SEQ pseudo instruction is required, containing the appropriate starting numbers. For example, in Figure 10, if the ten rates of the table were to be stored sequentially on the drum beginning at location 0045, the proper entries would be as shown in Figure 12.

TP	SN	LOCATION	OPER. CODE	DATA ADDRESS	TAG	INSTR. ADDRESS	TAG	REMARKS	ACCU UPPER 8003
1		INITI	ALI	ZATIO	N				
			- - -	- - - -					
			BLR	0000		0005			
			- - -	- - - -					
1		CALLI	NG	SEQUE	NC	E			
			RALK						
			LDD	EXIT		KFACT			
1		SUBRO	UTI	NE	CA	LC K	F	ACTORIAL	
		KFACT	SET	9000					
			LIB	0000		9000			
		0000	SEQ			9001			
			STD	9006					
			RAA	8002					
			SXA	0001					
			NZA	9004		9006			
			RAU	8002					
			MPY	8005		9002			
			SEQ						

Assigned Locations :

L		D	I
0000	24	9006	9001
0001	80	8002	9002
0002	51	0001	9003
0003	40	9004	9006
0004	60	8002	9005
0005	19	8005	9002

Figure 11. Using SEQ to Assign L- and I-Addresses of Subroutine

TP	SN	LOCATION	OPER. CODE	DATA ADDRESS	TAG	INSTR. ADDRESS	TAG	REMARKS	ACCU UPPER 8003
			- - -	- - - -		- - - -			
			BLR	0045		0056		RATE TABLE	
		0045	SEQ					STORE	
			00	0000		0126		TABLE	
			00	0000		0137			
			00	0000		0148			
		0050	SEQ						
			00	0000		0160			
			00	0000		0173			
			00	0000		0187			
			00	0000		0202			
			00	0000		0218			
			00	0000		0235			
			00	0000		0253			
			SEQ						
1									
		START	RCD	1950				READ CARD	
			- - -	- - - - -				- - - - -	

Assembled Table With Assigned L-Addresses

L	OP	D	I
	(---- TABLE ----)		
0045	00	0000	0126
0046	00	0000	0137
0047	00	0000	0148
0050	00	0000	0160
0051	00	0000	0173
0052	00	0000	0187
0053	00	0000	0202
0054	00	0000	0218
0055	00	0000	0235
0056	00	0000	0253

Figure 12. Using SEQ to Assign Locations in More than One Band

TAP (SEARCH LIBRARY TAPE)

This pseudo instruction can be used when assembling a symbolic program using symbolic routines (programs, subroutines) previously stored on magnetic tape. The magnetic tape prepared for such use, sometimes called a library tape, is described under *SOAP Librarian*. The format of the TAP pseudo instruction is shown in Figure 13.

The entry in the L-address part of the TAP instruction is the location of the first program instruction (that is, the entry point) of the routine to be called in from tape. The location entry may be in symbolic, regional, or absolute form.

The TAP pseudo instruction is placed in a symbolic source program at the point where a specific routine stored on a library tape is to be incorporated in the assembled output.

The assembly program, when it encounters a TAP pseudo instruction, searches the library tape for the routine entry point. When the entry point is located, the corresponding routine is read from tape onto the drum. The assembly program processes the called routine, producing an output TAP card followed by assembled output cards from the called routine. The assembly program continues assembling the source program, beginning with the symbolic instruction following the TAP pseudo instruction. Figure 14 shows the format of the output TAP card. Figure 15 is an excerpt from a symbolic program using TAP pseudo instructions to call in tape-stored subroutines at assembly time. Figure 16 is a listing of the symbolic card output produced from the library tape-to-card transfer; it represents the contents of the library tape. In Figure 16 the line of numerical information on the listing that precedes the first instruction and the line that follows the last instruction have no significance in each routine, except to indicate the beginning and the end of each routine. Figure 17 is a listing of the assembled output produced, using both the program of Figure 15 and the library tape represented by Figure 16 as input to the SOAP 2L Tape assembly program.

BD0 Operation Code

BD0 is not considered a valid symbolic operation code for *Branch on 8 in (Distributor) Position 10* in SOAP II, SOAP IIA, and Tape SOAP IIA (see footnote on page 24, *SOAP II Reference Manual*). However, both BDO and BD0 are valid symbolic operation codes in SOAP 2L and SOAP 2L Tape, and are assembled as numerical operation code 90.

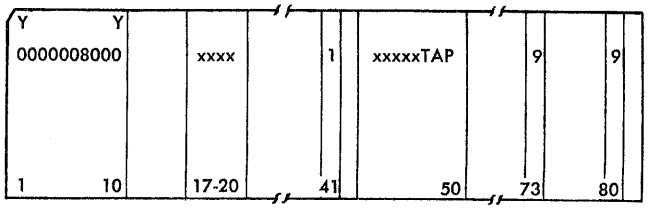


Figure 14. Format of an Output TAP Card

41	42	43 44- -47	48	49,50	51 52- -55		56	57- -60		61- -72	SOAP I COL.
41	42	43 44- -47	48	49,50	51 52- -55	56	57	58- -61	62	63- -72	SOAP II COL.
T P	S N	LOCATION	OPER. CODE		DATA ADDRESS	TAG		INSTR. ADDRESS	TAG	REMARKS	ACCU UPPER 8003
			- - -		- - - -						
		XXXXX	TAP		(blank)			(blank)			
			- - -		- - - -						

Figure 13. Format of TAP Pseudo Instruction

41	42	43	44- -47	48	49,50	51	52- -55		56	57- -60		61- -72	SOAP I COL.
41	42	43	44- -47	48	49,50	51	52- -55	56	57	58- -61	62	63- -72	SOAP II COL
T P	S N	LOCATION		OPER. CODE		DATA ADDRESS		TAG		INSTR. ADDRESS	TAG	REMARKS	ACCU UPPER 8003
				- - -		- - - -							
				L D D						U N P C K			
				S T L		Y 0 0 0 4				B G R A B			
		S T L 1		T A P									
		B G R A B		R A L		B B B B							
				D I V		7 I							
				R S L		8 0 0 3						N E G R E M N D R	
				A L O		B B B B							
				L D D						Z 0 0 0 0			
				S L O		9 1 0 0 I							
				B M I						E R R B			
				A L O		8 0 0 1				P A C K R			
		U N P C K		T A P									
		Z 0 0 0 0		T A P									
		9 1 0 0 I			0 0		0 0 0 0			9 1 0 0			
		- - - - -		- - -		- - - -				- - - -			

Figure 15. Excerpt from a Symbolic Program Using TAP Pseudo Operations

Literal Constant

Programs quite commonly make use of numerical constants. A 650 program written in symbolic language for assembly using SOAP II might contain instructions involving a numerical constant as indicated in Figure 18. The first symbolic instruction in Figure 18 referring to the constant 0000000001+ requires a separate entry on the program sheet (and a separate card punched from it) to store the constant 0000000001+ in symbolic location ONE.

The concept of the *literal constant,* however, not only provides for a symbolic program operation using a *constant,* but also produces a type-1 *literal* card. This card, when loaded, causes the constant to be stored in a symbolic location as well, provided the *literal-constant* symbol has not been previously defined. The preceding action results from one instruction.

Use literal constants in a program *initially* as follows. Specify the symbolic name of the constant in the D-address part of the instruction. Specify the numerical value of the constant in the *remarks* field of the instruction. Each of these two entries is explained subsequently.

D-ADDRESS, COLUMNS 51-55

Enter the symbolic name of the literal constant in this field of the instruction. Its form must be either bbXXX (for a positive constant) or bMXXX (for a negative constant), where X may be any valid character, including blank. However, there must be at least one non-blank character in columns 53-55.

REMARKS, COLUMNS 63-72

Enter the numerical value of the literal constant in this field of the instruction. The constant may be in the range 1 to 9999999999. The location within this field is immaterial. For example, if the constant 1 is being specified, it may be placed in any one column from 63 through 72, provided the remaining columns of the field are blank. Prior to being stored at loading-time in the symbolic location specified, the literal constant is right-justified.

```
     0000                    -0    -0 □3 □  □1□4 5      &□66 &□7
0000  2790501953  0000      81951 2790501955  1954    8012   5680121957  1956     8012  5580128000  1958
       37         Z0000     LDD   WORD2               LOAD CARD
       37                   STD   Y0002               ROUTINE
       37                   LDD   WORD4
       37                   STD   Y0004
       37                   LDD   WORD6
       37                   STD   Y0006
       37                   LDD   WORD5
       37                   STD   Y0005
       37                   LDD   WORD7
       37                   STD   Y0007
       37                   RAL   Z0001   STL1
0000  8990909091   0000      28373 7961730000  9100    0000   0000000000  0000     1956  0000000001  0037
       37         STL1      STL   Y0001
       37                   PCH   Y0001               INIT CNDTN
       37                   SLO   1I
       37                   NZE   STL1
       37                   HLT    0000   START       CDS READY
      280         WORD6      03    0000    0280       BEGIN INV
      280         WORD5      02    1000    0280       RECD NOW
      700         WORD4      03    0000    0700       RECD NEXT
     1702         WORD2      01    0000    1702       PER CONST
      500         WORD7      00    0000    0500       CASH AVAIL
        1         1I         00    0000    0001
6900  0090909090   0000      09090 0090900000  9091    0000   0000000000  0000     1956  0000000000  0037
       37         START     RCD   Z0000
       37                   RAL   Z0008
       37                   SRT    0005
       37                   DIV   250I
       37                   NZU   CDCOD
       37                   RAL   Z0009
       37                   SRT    0006
       37                   DIV   50I
       37                   NZU   CDCOD   FINE
     0000          700P      70    0000    0000
       37         CDCOD     RAL   700P    JMPIN
6463  9790767700   7664      17477 7961730000  6975    0000   0000000000  0000     1956  2419560037  0037
       37         UNPCK     RAL   Z0002
       37                   SLT    0002               RIGHT JUST
       37                   STU   MONTH
       37                   LDD   ZERO
       37                   SDA   NEEDA               DATA POSIT
       37                   SLO    8001
       37                   SLT    0003
       37                   STL   NEEDB   FLIP1       DATA POSIT
0000  7565656462   0000      67369 8283730000  7791    6183   7776826983  6100     1956  2419560037  0037
```

Figure 16. Symbolic Listing of Library Tape

Figure 19 shows several examples of entries on the 650 program sheet using literal constants. No program continuity is intended in these examples.

The assembled output of the first instruction would be:

```
   RAL  bbONE   1bbbbbbbbb  XXXX  65  YYYY  ZZZZ
1           LIT  1bbbbbbbbb  YYYY  00  0000  0001
```

The assembled output of the second instruction would be:

```
   ALO  bMFIC   14400bbbbb  XXXX  15  YYYY  ZZZZ
1           LIT  14400bbbbb  YYYY - 00  0001  4400
```

The assembled output of the third instruction would be:

```
   SUP  bb6bb   bbbbbbbbb6  XXXX  11  YYYY  ZZZZ
1           LIT  bbbbbbbbb6  YYYY  00  0000  0006
```

The assembled output of the last instruction would be:

```
   RSL  bMb2C   bbbb200bbb  XXXX  66  YYYY  ZZZZ
1           LIT  bbbb200bbb  YYYY - 00  0000  0200
```

Punching of literal constants is suspended during partial assembly and resumed in the multi-pass phase.

If the user reassembles a program that used literal symbols, causing generation of literal constant cards, new literal constant cards are generated and the previous literal cards are bypassed during reassembly.

Once a literal constant has been used initially as just described, subsequent operations using it can be performed by specifying the operation code and the symbolic name. The contents of the *remarks* field have no effect upon the literal constant. Because the remarks are reproduced in the output card, the field can be used for remarks in the usual manner. For example, if the first instruction shown in Figure 20 is the first use of the symbolic name, bb5bb (representing the literal constant 0000000005+), the constant, at program-loading time, will be stored in the location whose symbolic name is bb5bb and will be reset-added to 8002. This same literal constant can be used subsequently by an appropriate instruction, such as the second one in Figure 20. That instruction adds to 8002 the previously-defined constant, bb5bb, whose numerical value is

--			---	----			----	--	----	----
16			LDD		UNPCK		0043	69	0046	0099
17			STL	Y0004	BGRAB		0046	20	1930	0083
18	1			STL1 TAP						
19		STL1	STL	Y0001			0039	20	1927	0030
20			PCH	Y0001		INIT CNDTN	0030	71	1927	0027
21			SLO	1I			0027	16	0080	0035
22			NZE	STL1			0035	45	0039	0139
23			HLT	0000	START	CDS READY	0139	01	0000	0093
24		WORD6	03	0000	0280	BEGIN INV	0050	03	0000	0280
25		WORD5	02	1000	0280	RECD NOW	0100	02	1000	0280
26		WORD4	03	0000	0700	RECD NEXT	0150	03	0000	0700
27		WORD2	01	0000	1702	PER CONST	0200	01	0000	1702
28		WORD7	00	0000	0500	CASH AVAIL	0250	00	0000	0500
29		1I	00	0000	0001		0080	00	0000	0001
30		BGRAB	RAL	BBBB			0083	65	0017	0071
31			DIV	7I			0071	14	0074	0085
32			RSL	8003		NEG REMNDR	0085	66	8003	0143
33			ALO	BBBB			0143	15	0017	0121
34			LDD		Z0000		0121	69	0124	1950
35			SLO	9100I			0124	16	0077	0081
36			BMI		ERRB		0081	46	0034	0135
37			ALO	8001	PACKR		0034	15	8001	0041
38	1			UNPCKTAP						
39		UNPCK	RAL	Z0002			0099	65	1952	0107
40			SLT	0002		RIGHT JUST	0107	35	0002	0113
41			STU	MONTH			0113	21	0018	0171
42			LDD	ZERO			0171	69	0010	0163
43			SDA	NEEDA		DATA POSIT	0163	22	0067	0070
44			SLO	8001			0070	16	8001	0127
45			SLT	0003			0127	35	0003	0185
46			STL	NEEDB	FLIP1	DATA POSIT	0185	20	0189	0042
47	1			Z0000TAP						
48		Z0000	LDD	WORD2		LOAD CARD	1950	69	0200	0003
49			STD	Y0002		ROUTINE	0003	24	1928	0131
50			LDD	WORD4			0131	69	0150	0053
51			STD	Y0004			0053	24	1930	0133
52			LDD	WORD6			0133	69	0050	0103
53			STD	Y0006			0103	24	1932	0235
54			LDD	WORD5			0235	69	0100	0153
55			STD	Y0005			0153	24	1931	0084
56			LDD	WORD7			0084	69	0250	0203
57			STD	Y0007			0203	24	1933	0086
58			RAL	Z0001	STL1		0086	65	1951	0039
59		9100I	00	0000	9100		0077	00	0000	9100
--		-----	---	-----			----	--	----	----

Figure 17. Assembled Output of Program Shown in Figure 15

41	42	43 44- -47	48 49,50	51 52- -55		56 57- -60		61- -72	SOAP I COL.
41	42	43 44- -47	48 49,50	51 52- -55	56	57 58- -61	62	63- -72	SOAP II COL.
TP	SN	LOCATION	OPER. CODE	DATA ADDRESS	TAG	INSTR. ADDRESS	TAG	REMARKS	ACCU UPPER 8003
			RAU	ONE		STX			
		STX							
		ONE	00	0000		0001		CONSTANT	

Figure 18. Typical Use of a Constant in a SOAP II Instruction

41	42	43	44- -47	48	49,50	51	52- -55		56	57- -60		61- -72	SOAP I COL.
41	42	43	44- -47	48	49,50	51	52- -55	56	57	58- -61	62	63- -72	SOAP II COL.
T P	S N	LOCATION		OPER. CODE		DATA ADDRESS		TAG	INSTR. ADDRESS		TAG	REMARKS	ACCU UPPER 8003
				R	A L	b	b O N E					1	
				A	L O	b	M F I C					1 4 4 0 0	
				S	U P	b	b 6 b b					6	
				R	S L	b	M b 2 C					2 0 0	

Figure 19. Using Literal Constants

41	42	43	44- -47	48	49,50	51	52- -55		56	57- -60		61- -72	SOAP I COL.
41	42	43	44- -47	48	49,50	51	52- -55	56	57	58- -61	62	63- -72	SOAP II COL.
T P	S N	LOCATION		OPER. CODE		DATA ADDRESS		TAG	INSTR. ADDRESS		TAG	REMARKS	ACCU UPPER 8003
				-	- -	-	- - -		-	- - -			
				R	A L	b	b 5 b b		X	X X		5	
				-	- -	-	- - -						
				A	L O	b	b 5 b b		X	X X X			
				-	- -	-	- - -						

Figure 20. First and Subsequent Use of a Literal Constant

41	42	43	44- -47	48	49.50	51	52- -55		56	57- -60		61- -72	S.O.A.P. I COL.
41	42	43	44- -47	48	49,50	51	52- -55	56	57	58- -61	62	63- -72	S.O.A.P. II COL.
T P	S N	LOCATION		OPER. CODE		DATA ADDRESS		TAG	INSTR. ADDRESS		TAG	REMARKS	ACCU UPPER 8003
				E	Q U	b	b 1 b b			0 0 1 1		1	
				E	Q U	b	M P I b		A	0 0 2 0		3 1 4 1 5 9	
				E	Q U	b	b 1 4 4		M	F I C A		1 4 4 0 0	

Figure 21. Literal Constants in Equivalence Expressions

0000000005+. Literal constants can be used in conjunction with the two predefining pseudo operations, EQU and SYN.

LITERAL CONSTANTS IN EQUIVALENCE (EQU) EXPRESSIONS

The literal constant symbol written in the D-address is assigned the equivalent of the expression written in the I-address. Also, the numerical value of the literal constant written in the *remarks* field is stored in the location represented by the I-address at program-loading time. The I-address can be absolute, regional, or symbolic. If the I-address is regional or symbolic, it must have been previously defined. The examples shown in Figure 21 illustrate the use of literal constants in equivalence expressions.

The first pseudo instruction in Figure 21 establishes an equivalence between the literal constant symbol bb1bb and the absolute address 0011. It punches a literal card which, when loaded, stores the literal constant 0000000001+ in location 0011.

The second pseudo instruction in Figure 21 establishes an equivalence between the literal constant symbol bMPIb and the regional address A0020. It punches a literal card which, when loaded, stores the literal constant 0000314159— in the location whose regional address is A0020. Region A must have been previously defined by an REG card.

The third expression in Figure 21 establishes an equivalence between the literal constant symbol bb144 and the symbolic address MFICA. It punches a literal card which, when loaded, stores the literal constant 0000014400+ in the location whose symbolic address is MFICA. The symbolic address MFICA must have been previously defined.

41	42	43	44- -47	48	49.50	51	52- -55		56	57- -60		61- -72	S.O.A.P. I COL.
41	42	43	44- -47	48	49,50	51	52- -55	56	57	58- -61	62	63- -72	S.O.A.P. II COL.
T P	S N	LOCATION		OPER. CODE		DATA ADDRESS		TAG	INSTR. ADDRESS		TAG	REMARKS	ACCU UPPER 8003
				S	YN	b	bTWO		b	1304		2	
				S	YN	b	MEPS		E	PSIL		271828	
				S	YN	b	b9bb		T	0015		9999990001	

Figure 22. Literal Constants in Synonym Expressions

LITERAL CONSTANTS IN SYNONYM (SYN) EXPRESSIONS

An SYN card is like an EQU card except that the equivalent of the expression written in the I-address must be a drum address. This drum location is made unavailable to the program.

The first pseudo instruction in Figure 22 establishes an equivalence between the literal constant symbol bbTWO and the drum location 1304. It punches a literal card which, when loaded, stores the literal constant 0000000002+ in location 1304 and makes location 1304 unavailable to the program.

The second expression in Figure 22 establishes an equivalence between the literal constant symbol bMEPS and the symbolic address EPSIL. It punches a literal card which, when loaded, stores the literal constant 0000271828— in the drum location whose symbolic address is EPSIL, and makes the latter drum location unavailable to the program. The symbolic address EPSIL must have been previously defined.

The third pseudo instruction in Figure 22 establishes an equivalence between the literal constant symbol bb9bb and the regional drum address T0015. It punches a literal card which, when loaded, stores the literal constant 9999990001+ in the drum location whose regional address is T0015, and makes this latter location unavailable to the program. The regional address T0015 must have been previously defined by an REG card.

	Symbolic Input			Assembled Output Using SOAP II, IIA, IIA Tape				Assembled Output Using SOAP 2L, 2L Tape			
1802	RSL	9049C	NEXXT	1802	66	9649	0009	1802	66	9649	0011

Figure 23. Result of Modification in Optimization

Modification in Optimization — Special Case

SOAP 2L and SOAP 2L Tape have been improved in the case of an instruction having an A-, B-, or C-tagged IAS D-address and an *even* dynamic drum L-address. The improvement concerns the optimal location assigned to the instruction I-address. The change is in the assembly programs only, and results in proper optimization in this particular case. When comparing the I-addresses assigned using the SOAP 2L program with those assigned using the SOAP II and SOAP IIA programs, the saving is almost one drum revolution, as shown in Figure 23.

Regional Addresses

SOAP 2L and SOAP 2L Tape accept regional definitions with FWA > 1999. They will not make incorrect entries in the availability table, thereby permitting regionalization of tape unit, indexing register, and IAS addresses. (See Figure 24.)

41	42	43	44- -47	48	49.50	51	52- -55		56	57- -60		61- -72	S.O.A.P. I COL.
41	42	43	44- -47	48	49,50	51	52- -55	56	57	58- -61	62	63- -72	S.O.A.P. II COL.
T P	S N	LOCATION		OPER. CODE		DATA ADDRESS		TAG	INSTR. ADDRESS		TAG	REMARKS	ACCU UPPER 8003
				R	EG	X	8005			8007			
				R	EG	T	8010			8015			
				R	EG	I	9000			9059			

Figure 24. Regional Addresses for Indexing Registers, Tape Units, and IAS

SOAP 2L and SOAP 2L Tape accept 9000-9199, inclusive, as valid IAS addresses. The addresses 9060-9199, therefore, can be used in a program to indicate IAS addresses which are modified by program steps before execution of the instructions involved (for example, timing-ring settings for variable-length records).

SOAP Librarian

This two-part program is designed to:

1. prepare a library tape, that is, a magnetic tape containing routines that are frequently needed as subordinate parts of other routines or programs.
2. provide for punching the library-tape contents into symbolic cards.

The routines of a library tape are in symbolic form so that they can be assembled with symbolic programs into which the library routines are to be incorporated. See *TAP* under *Additional Pseudo Instruction; SOAP 2L and SOAP 2L Tape;* for a description of the use of library-tape routines with symbolic programs. The format of a library tape is indicated in Figure 25. The tape unit used to write a library tape must be 8012.

The first part of the SOAP Librarian program writes the library tape from SOAP II input cards. The second part punches the contents of the library tape into SOAP II output cards.

The format of each instruction on tape is the same as that of each instruction in the input area of the drum (see Figure 3 in Part II of the *SOAP II Reference Manual).* In performing a library-tape search for a routine, the part of the instruction format that has significance is word 1 (the L-address part of a symbolic instruction) of the first instruction in each routine.

DESCRIPTION OF SOAP LIBRARIAN PROGRAM DECK

The program deck supplied by IBM consists of 30 numbered cards, identified by end-printing, described as follows.

- Cards number 1-27 are self-loading program cards for both parts of the SOAP Librarian.
- Card number 28 is a transfer card that automatically transfers program control to begin reading input symbolic (SOAP II) cards.
- Card number 29 is the tape labeler card. It is needed only when creating a library tape but not when adding routines to an existing library tape.
- Card number 30 is a transfer card that transfers program control to the first instruction of the second (the tape-to-card) part of the SOAP Librarian. This card is used if, after creating a library tape or after making additions to it, the user wishes to perform a tape-to-card transfer and list the symbolic output. The listing is a printed reference record of the library tape.

GENERAL LIBRARY TAPE OPERATION

Referring to Figure 25 showing the (schematic) library-tape format, a search for a given library routine always begins with the library tape positioned at its load point. The pseudo operation TAP (see *Additional Pseudo Instructions; SOAP 2L and SOAP 2L Tape)* starts a tape search. A read-tape-check operation is executed, during which time tape is scanned until a tape mark is encountered. Upon reading a tape mark, the program checks the succeeding instruction (which is always the first instruction of a routine) to determine if its L-address contains the entry point being sought. If the entry point of the first instruction is not the one being sought, the program executes another read-tape-check operation, passing over the remaining instructions of the routine. Upon reading the next tape mark, the program checks the succeeding instruction as before, to determine if the L-address contains the entry point being sought. If it does not, the process is repeated until an L-address of the first instruction following a tape mark is found that agrees with the entry point being sought. The library routine thus located is written from tape onto the drum. The tape mark following the located library routine initiates a tape-rewind operation, leaving the library tape positioned at load point, ready for the next TAP pseudo operation.

In the event that a search of the library tape does not locate a routine, invalid information read beyond the double tape mark (that is, beyond the end of the library routine area) results in a storage-selection stop. To proceed with assembly by reading the next input card, manually transfer the program to drum location 1950.

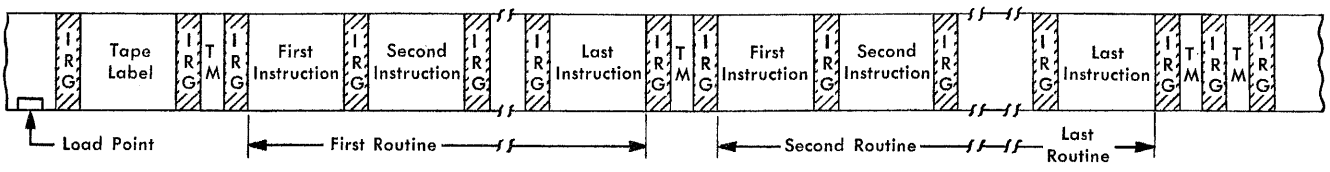


Figure 25. Library Tape Schematic

INPUT DECK FOR WRITING A LIBRARY TAPE

To write a library tape, place in the card reader an input card deck, arranged in the following order:

29—SOAP Librarian cards number 1-29.
X—Symbolic (SOAP II) program cards for the first library routine.
1—Card with a 12-punch in column 2; other columns blank.
X—Symbolic (SOAP II) program cards for the second library routine.
1—Card with a 12-punch in column 2; other columns blank.
.
. (Intervening routines in same manner.)
.
X—Symbolic (SOAP II) program cards for the last library routine.
1—Card with a 12-punch in column 2; other columns blank.
1—SOAP Librarian card number 30 (if a listing is desired of the library tape-to-card output).

ADDING ROUTINES TO AN EXISTING LIBRARY TAPE

For adding routines to an existing library tape, the input deck used, with one exception, is the same as that used for writing a library tape. The exception is that card number 30, the tape labeler card, is not used.

The SOAP Librarian program checks the entry point of each existing routine on tape to assure that the entry point of the routine being added is unique. If it is not unique, and if the *routines* are different, a unique entry point should be assigned to the added routine and punched in the L-address field (columns 43-47) of the first routine instruction.

After the SOAP Librarian program checks the entry point of the last existing routine in the manner indicated above, it encounters the double tape mark at the end of the existing library-routine area. It then backspaces one tape record (the last tape mark), erases it, and writes the instructions of the routine being added. In this manner all of the routines to be added are processed. After the last routine has been added, an additional tape mark is written, so that two consecutive tape marks indicate the end of the routine area.

Special Character Regions

In programs assembled by SOAP II, SOAP IIA, and SOAP IIA Tape, a regional address has the form ANNNN. A (representing the symbolizer part of the regional address) must be one of the 26 alphabetic characters and NNNN (representing the absolute part of the regional address) must be a four-digit number. Using SOAP 2L or 2L Tape in programs assembled on an IBM 650 equipped with the group II special character device extends the characters that can be used in the symbolizer part of a regional address to include the eleven (group II) special characters, as well as the 26 alphabetic characters. For example, if regions R, D, $, and # are defined as locations 1951-1960, 1100-1129, 0401-0406, and 0963-0969, respectively, the regional addresses are assembled as indicated:

R0008	1958
D0001	1100
$0003	0403
#0007	0969

NOTE: On IBM 407 Accounting Machines equipped with FORTRAN characters, both the @ and the — characters print as —. However, a unique address is assigned to each regional address that uses @ and — as the symbolizer part, even if the absolute part is the same. Thus, if region @ has been defined as locations 1727-1736 and region — has been defined as locations 1901-1912, then the regional addresses @0003 and —0003, while both print on a 407 equipped with FORTRAN characters as —0003, are assigned locations 1729 and 1903, respectively.

Special Operation Codes

The following positive special operation codes can be assembled.

Table Look Up on Equal	TLE	+63
Read Sorter Reader	RSR	+12

The following negative special operation codes can be assembled.

Alpha-to-Numeric Conversion	ANC	—20
Numeric-to-Alpha Conversion	NAC	—65
Read Tape Special	RTS	—05
Set Format	SFM	—19
Special Shift Instruction	SPS	—30
Typewriter Output	TYO	—79
Write Tape Special	WTS	—07
Not Equal Upper	NEU	—42
Equal Low Upper	ELU	—43

SOAP 2L and SOAP 2L Tape programs translate the preceding negative-operation-code mnemonics into the proper numerical codes and automatically make the instructions negative. Be careful, therefore, when modifying an instruction using a negative operation code.

For example, let —ABC represent any of the preceding negative operation codes in the instruction —ABC R0006 NEXT, located in INSTR. Suppose you want to change the data address of this instruction to R0007. If the constant 0000010000+ were added to the contents of INSTR, the resultant data address would be R0005 and not the desired R0007. The correct procedure is either to add 0000010000— or to subtract 0000010000+.

Be careful also not to alter the sign of an instruction if the instruction would then be interpreted differently by the IBM 650. If during the course of address modification a SET FORMAT instruction were made positive and executed, the resultant operation would be a MULTIPLY (19) instruction. Conversely, if an RAL instruction were made negative, the effective operation would become NAC.

If a negative instruction is to be modified by the contents of an indexing register, the sign of the instruction need not be considered. The 650 treats all instructions as if they were positive in sign while they are being indexed.

Type 3 Cards

Any type *blank* card, i.e., any card other than a *comments* (type 1) card or *relocatable* (type 2) card, can be specified as a type-3 card by a 3-punch in column 41. The 3-punch does not affect assembly; it appears in the output card, together with 69 1954 8000 in word 1, columns 1-10. When the assembled output is loaded, type-3 cards are bypassed. Therefore, type-3 cards in assembled output need not be removed from the deck prior to loading.

When coding an instruction that is modified during program operation, using type-3 cards permits optimizing the D- or the I-addresses. For example, in Figure 26, the instruction located in 2WAYS is one of the two instructions indicated by the two type-3 cards, depending on a program test or decision. It is desirable to optimize the locations ZOOM1, DLAXX, and ZOOM2. Using the two type-3 cards as indicated accomplishes this.

Note two significant points regarding the use of type-3 cards:

1. They appear in the symbolic program deck prior to the instructions they represent.
2. They clarify the program executive listing.

IBM 533 Control Panel

The SOAP II control panel for the 533 cannot be used with SOAP 2L or SOAP 2L Tape. The reason is that assembly stops with a branch distributor operation code in the program register as soon as the first symbol is encountered. The SOAP II control panel for the 533 can, however, be modified so that it can be used with the SOAP 2L and SOAP 2L Tape programs. On the other hand, the SOAP 2L-SOAP 2L Tape control panel described in the following paragraph *can* be used with the SOAP II program. When this procedure is used, all output cards except availability-table cards will contain a zero punch in column 80. This punch, however, does hinder subsequent reassembly of the output deck and can be disregarded.

A 533 control panel for SOAP 2L-SOAP 2L Tape can be wired from a SOAP II 533 panel by the following steps.

A. Remove from the SOAP II panel:
   1. All wires from READ CARD B to words 7, 8, and 9 of STORAGE ENTRY C.
   2. The wire from the WORD SIZE EMITTER 3 to word 10 of WORD SIZE ENTRY C.
   3. The wire from column 41 of FIRST READING to the left READ DIGIT SELECTOR and to D PU of PILOT SELECTOR 1.
   4. The wire from the TRANSFER of PILOT SELECTOR 7 to column 79 of PUNCH CARD A.

B. Add to the remaining wiring of the SOAP II panel the wiring shown in Figure 27.

If a new panel is to be wired for SOAP 2L or SOAP 2L Tape, the panel should be wired as shown in *Figure* 8 of the *SOAP II Reference Manual*, omitting the wiring just enumerated. The wiring shown in Figure 27 should then be added.

IBM 543 and 544 Wiring Lists

Figures 28 and 29, respectively, are wiring lists for the IBM 543 Card Reader and the IBM 544 Card Punch.

Operating Instructions

Figures 30 and 31, respectively, are the operating instructions for SOAP 2L and SOAP 2L Tape.

41	42	43	44- -47	48	49.50	51	52- -55		56	57- -60		61- -72	S.O.A.P. I COL.
41	42	43	44- -47	48	49,50	51	52- -55	56	57	58- -61	62	63- -72	S.O.A.P. II COL.
T P	S N	LOCATION		OPER. CODE		DATA ADDRESS		TAG	INSTR. ADDRESS		TAG	REMARKS	ACCU UPPER 8003
				N	Z E	2	W A Y S		I	L L O P			
3		2	W A Y S	S	L T		0 0 0 2		Z	O O M 1			
3		2	W A Y S	S	L O	D	L A X X		Z	O O M 2			

Figure 26. Type-3 Instructions

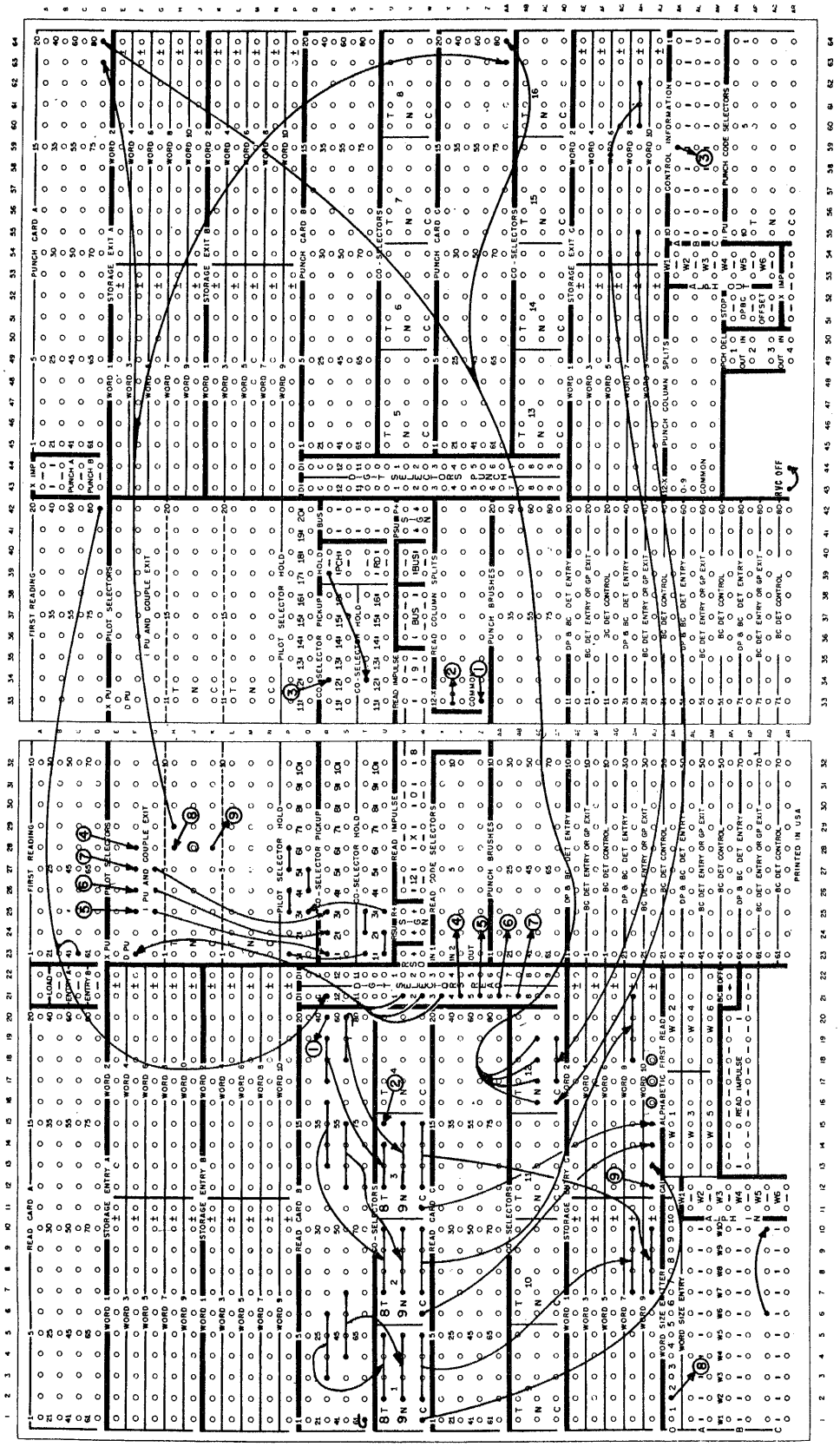


Figure 27. Additional 533 Control-Panel Wiring for SOAP

NOTE: All unlabeled multiple components are considered to be numbered left to right (e.g., co-selector and pilot-selector positions, etc.). The only exceptions are word-entry digits which are numbered right to left.

From	*To*
First Read: 2	Load FMT
First Read: 43-47	Alpha Pre Read Wd 1: 1-5
First Read: 48-50	Alpha Pre Read Wd 4: 1-3
First Read: 51-55	Alpha Pre Read Wd 2: 1-5
First Read: 56	Alpha Pre Read Wd 4: 4
First Read: 57-61	Alpha Pre Read Wd 3: 1-5
First Read: 62	Alpha Pre Read Wd 4: 5
First Read: 63-67	Alpha Pre Read Wd 5: 1-5
First Read: 68-72	Alpha Pre Read Wd 6: 1-5
First Read: 41	Left Digit Selector Common
First Read: 42	Pilot Selector 2 DPU
First Read: 80	Left Digit Selector Common
Left Digit Selector 12	Load FMT
Left Digit Selector 1-3	Pilot Selector 7 DPU
Left Digit Selector 4	Pilot Selector 6 DPU
Left Digit Selector 5	Pilot Selector 3 DPU
Left Digit Selector 6	Pilot Selector 4 DPU
Left Digit Selector 7	Pilot Selector 5 DPU
Co-Selector 5: T 1	Emit 8
Co-Selector 5: N 1	Emit 9
Co-Selector 5: C 1	Wd 10 Entry: 9
Co-Selector 5: C (2-5)	Wd 7 Entry: 4-1
Co-Selector 7: T 1	Emit 8
Co-Selector 7: N 1	Emit 9
Co-Selector 7: C 1	Wd 10 Entry: 8
Co-Selector 7: C (2-5)	Wd 8 Entry: 4-1
Co-Selector 8: T 1	Read Impulse: 8
Co-Selector 8: N 1	Read Impulse: 9
Co-Selector 8: C 1	Wd 10 Entry: 7
Co-Selector 8: C (2-5)	Wd 9 Entry: 4-1
Second Reading: 23-26	Co-Selector 5: T (2-5)
Second Reading: 33-36	Co-Selector 7: T (2-5)
Second Reading: 37-39	Co-Selector 8: T (2-4)

From	*To*
Second Reading: 40	Column Split 7 Common
Second Reading: 41	Top Pilot Selector 7 T
Second Reading: 44-47	Co-Selector 5: N (2-5)
Second Reading: 52-55	Co-Selector 7: N (2-5)
Second Reading: 58-61	Co-Selector 8: N (2-5)
Second Reading: 43-47	Wd 1 Entry: 5-1
Second Reading: 48-50	Wd 4 Entry: 5-3
Second Reading: 51-55	Wd 2 Entry: 5-1
Second Reading: 56	Wd 4 Entry: 2
Second Reading: 57-61	Wd 3 Entry: 5-1
Second Reading: 62	Wd 4 Entry: 1
Second Reading: 63-67	Wd 5 Entry: 5-1
Second Reading: 68-72	Wd 6 Entry: 5-1
Pilot Selector 2: T	Read Impulse: 8
Pilot Selector 2: N	Read Impulse: 0
Pilot Selector 2: C	Wd 10 Entry: 1
Pilot Selector 7: Top N	Emit 0
Pilot Selector 7: Top C	Wd 10 Entry: 3
Pilot Selector 7: Bottom T	Read Impulse: 8
Pilot Selector 7: Bottom N	Read Impulse: 0
Pilot Selector 7: Bottom C	Wd 10 Entry: 2
Pilot Selector 6: T	Emit 1
Pilot Selector 6: N	Emit 0
Pilot Selector 6: C	Wd 10 Entry: 10
Column Split 7: 0-9	Co-Selector 8: T 5
Wd 10 Entry: 6-4	Read Impulse: 0
Load FMT	Load D/I
Couple Exit P. S. 3	Co-Selector Pickup: 5
Couple Exit P. S. 4	Co-Selector Pickup: 7
Couple Exit P. S. 5	Co-Selector Pickup: 6
CAI, N/FMT	Alpha in Wds 1-6
Word Size Emitter: 4	Wd Size Entry: 7-9
RVC Off	(A Jackplug)

Figure 28. SOAP Wiring List for IBM 543

NOTE 1: All unlabeled multiple components are considered to be numbered left to right (e.g., co-selector and pilot selector positions, etc.). The only exceptions are column splits which are numbered top to bottom.

NOTE 2: Field-Selectors Level-2 (Card Type B) are wired to the top Punch Magnet Entries.
Field-Selectors Level-1 (Card Type A) and Field Selectors, Normal (Card Type C) are wired to the bottom Punch Magnet Entries.

From	*To*
Control Information 1	Co-Selector PU 4
Control Information 2	Co-Selector PU 1
Control Information 3	Pilot Selector PU 1, 4, 5
Control Information 4	Co-Selector PU 3
Control Information 5	Co-Selector PU 2
Control Information 6	Field Selector PU (1-6) Level 2
Control Information 7	Field Selector PU (1-6) Level 1
Control Information 8	Pilot Selector PU 3
Control Information 9	Pilot Selector PU 2
Control Information 10	Alpha Out Words (1-6) (SOAP 4000 only)
Common Field Selector 1: (1-10)	Word 1 Exit (1-10)
Common Field Selector 1: (11)	PCH +
Common Field Selector 1: (12-21)	Word 2 Exit: (1-10)
Common Field Selector 1: (22)	Emit 9
Common Field Selector 2: (1-10)	Word 3 Exit: (1-10)
Common Field Selector 2: (11)	Word 9 Exit: 7
Common Field Selector 3: (1-10)	Word 4 Exit: (1-10)
Common Field Selector 3: (11)	Word 9 Exit: 8
Common Field Selector 3: (13-21)	Word 5 Exit: (2-10)
Common Field Selector 3: (22)	Word 9 Exit 9:
Common Field Selector 4: (1-10)	Word 6 Exit: (1-10)
Common Field Selector 4: (11)	PSU
Common Field Selector 4: (12-21)	Word 7 Exit: (1-10)
Common Field Selector 4: (22)	Top Pilot Selector 3 C
Common Field Selector 5: (1-10)	Word 8 Exit: (1-10)
Common Field Selector 5: (11)	Word 9 Exit: 10
CAI	Alpha Out (Except SOAP 4000)
Common Field Selector 6: (1-10)	Punch Magnet Entry (1-10)
Common Field Selector 6: (11)	Column Split 12 Common
Common Field Selector 6: (12-18)	Punch Magnet Entry (21-27)
Common Field Selector 6: (19)	Punch Magnet Entry 29
Common Field Selector 6: (20)	Punch Magnet Entry 76
Common Field Selector 6: (21)	Bottom P. S. 2 C
Common Field Selector 6: (22)	Punch Magnet Entry 29
Level 1 Field Selector 1: (6-10)	Punch Magnet Entry (43-47)
Level 1 Field Selector 1: (17-21)	Punch Magnet Entry (51-55)
Level 1 Field Selector 1: (22)	Punch Magnet Entry 73
Level 1 Field Selector 2: (6-10)	Punch Magnet Entry (57-61)
Level 1 Field Selector 2: (11)	Punch Magnet Entry 17
Level 1 Field Selector 3: (6-8)	Punch Magnet Entry (48-50)
Level 1 Field Selector 3: (9)	Punch Magnet Entry 56
Level 1 Field Selector 3: (10)	Punch Magnet Entry 62
Level 1 Field Selector 3: (11)	Punch Magnet Entry 18
Level 1 Field Selector 3: (17-21)	Punch Magnet Entry (63-67)
Level 1 Field Selector 3: (22)	Punch Magnet Entry 19
Level 1 Field Selector 4: (6-10)	Punch Magnet Entry (68-72)
Level 1 Field Selector 4: (22)	Punch Magnet Entry 79
Level 1 Field Selector 5: (2)	Punch Magnet Entry 80
Level 1 Field Selector 5: (7-9)	Punch Magnet Entry 80
Level 1 Field Selector 5: (10)	Punch Magnet Entry 41
Level 1 Field Selector 5: (11)	Punch Magnet Entry 20
Level 1 Field Selector 6: (1)	Column Split 12 Common
Level 1 Field Selector 6: (2-6)	Emit 0
Level 1 Field Selector 6: (7)	Emit 8
Level 1 Field Selector 6: (8-9)	Emit 0
Level 1 Field Selector 6: (10)	Column Split 12 Common

From	*To*
Level 2 Field Selector 1 (1-10)	Punch Magnet Entry (1-10)
Level 2 Field Selector 1: (11)	PCH +
Level 2 Field Selector 1: (12-21)	Punch Magnet Entry (11-20)
Level 2 Field Selector 2: (1-10)	Punch Magnet Entry (21-30)
Level 2 Field Selector 3: (1-10)	Punch Magnet Entry (31-40)
Level 2 Field Selector 3: (12-21)	Punch Magnet Entry (41-50)
Level 2 Field Selector 4: (1-10)	Punch Magnet Entry (51-60)
Level 2 Field Selector 4: (11)	PSU
Level 2 Field Selector 4: (12-21)	Punch Magnet Entry (61-70)
Level 2 Field Selector 5: (1-10)	Punch Magnet Entry (71-80)
Level 2 Field Selector 6: (11)	Column Split 11: (11-12)
Normal Field Selector 1: (6-10)	Punch Magnet Entry (43-47)
Normal Field Selector 1: (17-21)	Punch Magnet Entry (51-55)
Normal Field Selector 1: (22)	Field Selector 6: 2 Common
Normal Field Selector 2: (6-10)	Punch Magnet Entry (57-61)
Normal Field Selector 2: (11)	Punch Magnet Entry (17)
Normal Field Selector 3: (6-8)	Punch Magnet Entry (48-50)
Normal Field Selector 3: (9)	Punch Magnet Entry 56
Normal Field Selector 3: (10)	Punch Magnet Entry 62
Normal Field Selector 3: (11)	Punch Magnet Entry 18
Normal Field Selector 3: (17-21)	Punch Magnet Entry (63-67)
Normal Field Selector 3: (22)	Punch Magnet Entry 19
Normal Field Selector 4: (6-10)	Punch Magnet Entry (68-72)
Normal Field Selector 4: (12-13)	Co-Selector 4: C (4-5)
Normal Field Selector 4: (14)	Top Pilot Selector 4 N
Normal Field Selector 4: (15)	Bottom Pilot Selector 4 N
Normal Field Selector 4: (16)	Top Pilot Selector 5 N
Normal Field Selector 4: (17)	Bottom Pilot Selector 5 N
Normal Field Selector 4: (18-20)	Co-Selector 1: N (1-3)
Normal Field Selector 4: (21)	Column Split 8: (0-9)
Normal Field Selector 4: (22)	Punch Magnet Entry 79
Normal Field Selector 5: (2)	Punch Magnet Entry 80
Normal Field Selector 5: (3-6)	Co-Selector 3: N (1-4)
Normal Field Selector 5: (7-9)	Punch Magnet Entry 80
Normal Field Selector 5: (10)	Punch Magnet Entry 41
Normal Field Selector 5: (11)	Punch Magnet Entry 20
Normal Field Selector 6: (1)	Column Split 10 Common
Normal Field Selector 6: (2)	Field Selector 1: N 22
Normal Field Selector 6: (3)	Emit 1
Normal Field Selector 6: (4)	Punch Magnet Entry 2
Normal Field Selector 6: (5)	Emit 5
Normal Field Selector 6: (6)	Emit 0
Normal Field Selector 6: (7-9)	Co-Selector 2: C (2-4)
Normal Field Selector 6: (10)	Punch Column Split 9 Common
Normal Field Selector 6: (11)	Punch Magnet Entry 30
Normal Field Selector 6: (12)	Emit 2
Normal Field Selector 6: (13)	Emit 4
Normal Field Selector 6: (14-17)	Co-Selector 3: C (1-4)
Normal Field Selector 6: (18)	Emit 8
Normal Field Selector 6: (19)	Emit 0
Normal Field Selector 6: (20)	Co-Selector 4: C 3
Normal Field Selector 6: (21)	Column Split 8: (11-12)
Normal Field Selector 6: (22)	Punch Magnet Entry 28
D I	Top Digit Selector (Emitter)
Top Digit Selector (Emitter) 12	Bottom Pilot Selector 2 N
Top Digit Selector (Emitter) 12	Column Split 9: (11-12)

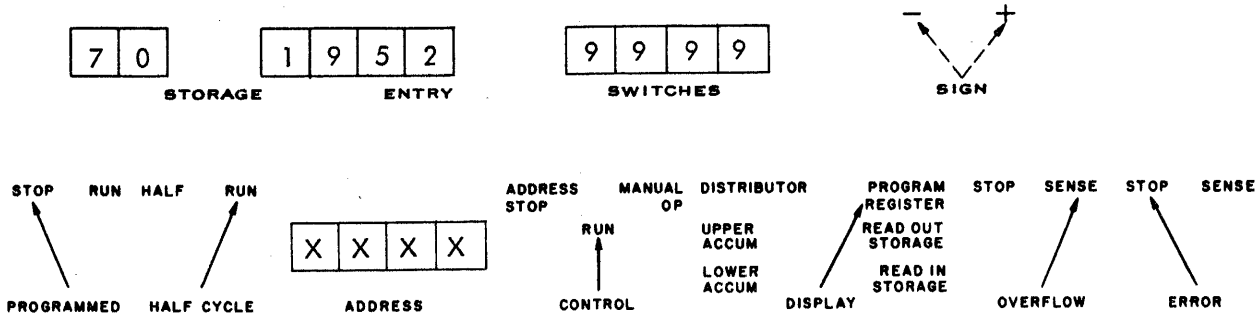
Figure 29 (1). SOAP Wiring List for IBM 544

From	*To*
Top Digit Selector (Emitter) 0	Co-Selector 2 T 5
Top Digit Selector (Emitter) 9	Co-Selector 2 N 3
Digit Emitter 12	Column Split 10: (11-12)
Digit Emitter 12	Column Split 12: (11-12)
Digit Emitter 11	Top Pilot Selector 2 T
Digit Emitter 11	Co-Selector 3 T 5
Digit Emitter 11	Co-Selector 4 T 3
Digit Emitter 11	Co-Selector 2 T 1
Digit Emitter 0	Co-Selector 2 T 3
Digit Emitter 0	Co-Selector 2 T 4
Digit Emitter 0	Column Split 12: (0-9)
Digit Emitter 1	Co-Selector 2 N 2
Digit Emitter 3	Co-Selector 2 N 5
Digit Emitter 5	Co-Selector 2 N 4
Digit Emitter 6	Column Split 10: (0-9)
Digit Emitter 8	Co-Selector 2 T 2
Digit Emitter 9	Top Pilot Selector 3 T
Pilot Selector 1 PU	Pilot Selector 4 PU
Pilot Selector 4 PU	Pilot Selector 5 PU

From	*To*
Top Pilot Selector 1 T	X Impulse
Top Pilot Selector 1 C	Punch Magnet Entry 77
Top Pilot Selector 2 C	Punch Magnet Entry 42
Bottom Pilot Selector 2 T	X Impulse
Co-Selector 1: T 5	X Impulse
Co-Selector 1: N 4	Column Split 8: C
Co-Selector 1: C (1-4)	Punch Magnet Entry (37-40)
Co-Selector 1: C 5	Punch Magnet Entry 78
Co-Selector 2: C 1	Punch Magnet Entry 74
Co-Selector 2: C 5	Column Split 9: (0-9)
Co-Selector 3: C 5	Punch Magnet Entry 75
Co-Selector 4: N (4-5)	Punch Magnet Entry (31-32)
Run INLK	(A Jackplug)
Top Pilot Selector 4 C	Punch Magnet Entry 33
Bottom Pilot Selector 4 C	Punch Magnet Entry 34
Top Pilot Selector 5 C	Punch Magnet Entry 35
Bottom Pilot Selector 5 C	Punch Magnet Entry 36
Common Field Selector 3: 12	Column Split 11 Common
Word 5 Exit: 1	Column Split 11: (0-9)

Figure 29 (2). SOAP Wiring List for IBM 544

IBM 650 PROGRAM OPERATING INSTRUCTIONS



Initial Console Setting as shown above.

A. Normal Starting Procedure: Computer Reset; Program Start.

B. Special Instructions

If SOAP 2L is already on the drum do one of the following:

1. Set 00 0000 1000 in the Storage Entry switches.
2. Precede input with a B O P card and set 00 0000 1950 in the Storage Entry switches.

Set Sign switch to minus (-) if automatic symbol-table punch-out is desired.

Card Input - Output

READ FEED

NO. OF CARDS	FILE DESCRIPTION
296	SOAP 2L (including loader)
xxx	Symbolic program deck
1	PST card (if desired)

PUNCH FEED

CARD FORM
SOAP II

CONTROL PANELS

SOAP 2L control panel
for appropriate card Input-
Output components

TAPE UNITS

ADDRESS	INPUT, OUTPUT OR OTHER	FILE PROTECTION RING IN	FILE PROTECTION RING OUT	LABEL CHARACTERISTICS	FILE DESCRIPTION
8010					
8011					
8012	Library Tape		X	(Optional—used only if TAP pseudo instruction is used)	
8013					
8014					
8015					

Figure 30 (1). SOAP 2L Operating Instructions

Other Instructions and Remarks:

After each machine pass discard the last card out of the punch feed. If the preceding card contains a 9 punch in column 80, or if it is an availability-table card, assembly is complete. If the card does not meet either of these conditions, remove all cards that contain a 9 punch in column 80 from the deck. Use the remainder of the deck for input to the next pass. The first card that does not contain a 9 punch in column 80 (i.e., the first card of the input to the next pass) will contain a 4 punch in that column.

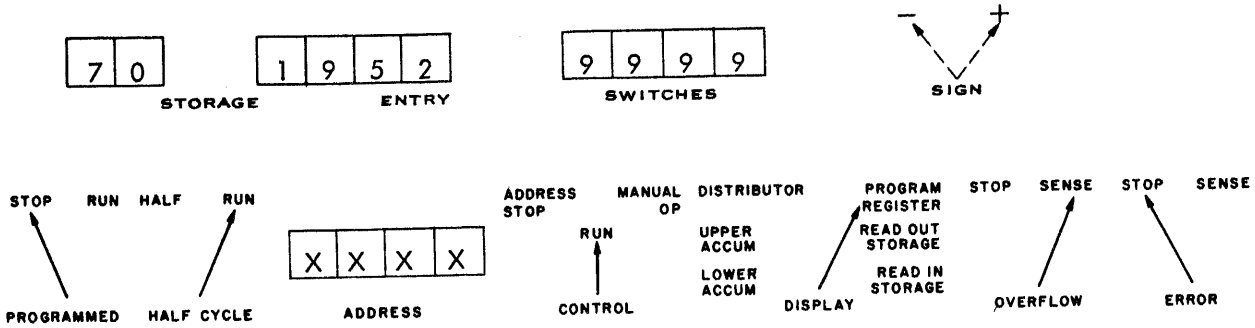
An availability table punch-out may be initiated manually by transferring to location 1900, and a symbol table punch-out by transferring to location 1800.

Program Stops and Required Action:

STOP ADDRESS	MESSAGE - EXPLANATION - ACTION
0222	No locations available for the remaining portion of the program being assembled. Press the Program-Start key to continue assembly. Addresses not assigned will be left blank in the output cards.

Figure 30 (2). SOAP 2L Operating Instructions

IBM 650 PROGRAM OPERATING INSTRUCTIONS



Initial Console Setting as shown above.

A. Normal Starting Procedure: Computer Reset; Program Start.

B. Special Instructions.

If SOAP 2 L Tape is already on the drum do one of the following:

1. Set 00 0000 1000 in the Storage Entry switches.
2. Precede input with a B O P card and set 00 0000 1950 in the Storage Entry switches.

Set Sign switch to minus (-) if automatic symbol-table punch-out is desired.

Card Input - Output

READ FEED

NO. OF CARDS	FILE DESCRIPTION
309	SOAP 2L Tape (including loader)
xxx	Symbolic program deck
1	PST card (if desired)
1	XYZ card

PUNCH FEED

CARD FORM
SOAP II

CONTROL PANELS

SOAP 2L control panel
for appropriate card Input-
Output components

TAPE UNITS

ADDRESS	INPUT, OUTPUT OR OTHER	FILE PROTECTION RING IN	FILE PROTECTION RING OUT	LABEL CHARACTERISTICS	FILE DESCRIPTION
8010	Input-output	X			Work tape
8011	Input-output	X			Work tape
8012	Library Tape		X	(Optional—used only if TAP pseudo instruction is used)	
8013					
8014					
8015					

Figure 31 (1). SOAP 2L Tape Operating Instructions

Other Instructions and Remarks:

Discard last output card.

An availability-table punch-out may be initiated manually by transferring to location 1900, and a symbol-table punch-out by transferring to location 1800.

Program Stops and Required Action:

STOP ADDRESS	MESSAGE - EXPLANATION - ACTION
0222	No locations available for the remaining portion of the program. Pressing the Program-Start key continues assembly. Addresses not assigned are left blank in the output cards.
0444	End-of-file detected while writing a tape record.* Replace tape with a longer one and begin assembly again.
0555	Error detected while writing a tape record.* Pressing the Program-Start key causes one attempt to rewrite the error record. Do not attempt to rewrite more than three times.
0666	Error detected while writing a tape mark.* Pressing the Program-Start key causes one attempt to rewrite the tape mark. Do not attempt to rewrite more than three times.
0777	Error detected while reading a tape record.* Pressing the Program-Start key causes as many as four attempts to reread. If the error condition persists, the record may be examined by a console read-out**, or may be printed out, if a 407 Accounting Machine is coupled to Synchronizer 2, by transferring to location 1550. If the record can be corrected manually from the console, transfer to location 1650 to resume assembly.
0888	Error print-out completed.
0999	End of job. Re-initialize before beginning assembly by preceding the program with a BOP card and pressing the Program Start key, or by transferring to location 1000.

* Tape unit address can be determined by displaying the contents of the lower accumulator.
** Records read from tape occupy locations 9050-9059. Format is similar to that shown in Figure 3, Part II of the SOAP II Reference Manual.

Figure 31 (2). SOAP 2L Tape Operating Instructions

SOAP 4000 and SOAP 42

Symbolic programs written for an IBM 650 Model 4 (4000-word drum) can be assembled on a 650 Model 4 by using the SOAP 4000 program. Symbolic programs written for a 650 Model 4 can be assembled on a 650 Model 2 (2000-word drum) by using the SOAP 42 program. Both programs include most of the program features of SOAP 2L and SOAP 2L Tape. The differences that exist are pointed out in this section of the bulletin.

Program Assembly

Translation and assembly of the symbolic program instructions are the same as explained for SOAP 2L and SOAP 2L Tape. The sizes of the symbol tables, however, are different.

Symbol Capacity

For SOAP 4000, the maximum number of symbols that can be entered into the symbol table is 998, 999, or 1000, depending on the number of symbols in the instruction being processed when the 998th symbol is entered. Similarly, for SOAP 42, the symbol-table capacity is 178, 179, or 180.

Availability-Table Punchout

The availability-table punchout for SOAP 4000 occurs in two sections. The first section represents the status of the first 2000 drum locations, while the second section represents the last 2000 drum locations.

Programming Features

The following programming features, except the SOAP Librarian, apply to both SOAP 4000 and SOAP 42. These features are explained under *Programming Features* in the section for SOAP 2L and SOAP 2L Tape.

- BD0 Operation Code
- Literal Constant
- Modification in Optimizing—Special Case
- SOAP Librarian (Not used with SOAP 42.)
- Special-Character Regions
- Special Operation Codes
- Type-3 Cards

Address Modification

For SOAP 4000 and SOAP 42, as for SOAP II, SOAP IIA, and Tape SOAP IIA, the user can tag an instruction in symbolic form using indexing register A, B, or C. Simply enter A, B, or C in the appropriate tag columns (columns 56 and 62) of the 650 SOAP Program Sheet. However, the increased range of drum addresses of a 4000-word 650 requires a change in the *method* of tagging instructions that use indexing registers. The following rules govern tagging of instructions to be assembled by SOAP 4000 and SOAP 42.

NORMAL OPERATION

1. All drum and IAS D-addresses meaningful to the operation code can be tagged.
2. 80xx (i.e., 8000-8003, 8005-8007, 8010-8015) D-addresses can be tagged by indexing register B *only*.
3. IAS I-addresses can be tagged, with the following exceptions:
   a. The corresponding D-addresses are B- or C-tagged drum addresses.
   b. The corresponding D-addresses are B-tagged 80xx addresses.
   c. The tags of the I-addresses use the same indexing registers as do the operation codes.

SPECIAL OPERATION (BY INTERNAL PROGRAMMING)

The assembly programs SOAP 4000 and SOAP 42 will process, by internal programming, instructions containing:

1. tagged drum I-addresses.
2. tagged IAS I-addresses that have B- or C-tagged drum D-addresses.
3. tagged IAS I-addresses that have B-tagged 80xx D-addresses.

The preceding occur on the condition that, if the operations are indexing-register arithmetic or indexing-register branch codes, the tags do not use the same indexing registers as do the operation codes. As indicated in examples that follow, tagging I-addresses of these instructions produces two output instructions at assembly time for each input tagged instruction.

Input symbolic instructions, tagged in accordance with the preceding rules, are automatically assembled in either normal or special operation. In normal operation they yield one assembled output instruction per input tagged instruction. In special operation they yield two assembled output instructions per input tagged instruction. Each of the two operations are subsequently described in detail.

NORMAL OPERATION—TAGGED DRUM AND IAS D-ADDRESS

Indexing Register A. SOAP 4000 and SOAP 42 programs process an A-tag of a drum D-address of a symbolic instruction by adding 4000 to the drum D-address at assembly time.

EXAMPLES:

Symbolic Input			*Assembled Output*		
RAL	3699A	1254	65	7699	1254
AXB	0001A	2534	52	4001	2534

SOAP 4000 and SOAP 42 programs process an A-tag of an IAS D-address by adding 200 to the IAS D-address at assembly time.

EXAMPLES:

Symbolic Input			*Assembled Output*		
RAU	9003A	0253	60	9203	0253
RAC	9019A	3823	88	9219	3823

Indexing Register B. SOAP 4000 and SOAP 42 programs process a B-tag of a drum or an 80xx D-address by adding either 4000 to the drum I-address or 800 to the 80xx or IAS I-address at assembly time.

EXAMPLES:

Symbolic Input			*Assembled Output*		
AUP	1035B	1142	10	1035	5142
AUP	1035B	8005	10	1035	8805
AUP	1035B	9006	10	1035	9806
ALO	8007B	3410	15	8007	7410
ALO	8007B	8005	15	8007	8805
ALO	8007B	9002	15	8007	9802

SOAP 4000 and SOAP 42 programs process a B-tag of an IAS D-address by adding 400 to the IAS D-address at assembly time.

EXAMPLES:

Symbolic Input			*Assembled Output*		
SLO	9040B	9002	16	9440	9002
SLO	9040B	2307	16	9440	2307
SLO	9040B	8003	16	9440	8003

Indexing Register C. SOAP 4000 and SOAP 42 programs process a C-tag of a drum D-address by adding to the drum D-address 4000 and to the I-address either 4000 (if the I-address is a drum location) or 800 (if the I-address is 80xx or an IAS location).

EXAMPLES:

Symbolic Input			*Assembled Output*		
RSU	0515C	1649	61	4515	5649
RSU	0515C	9027	61	4515	9827
RSU	0515C	8003	61	4515	8803

SOAP 4000 and SOAP 42 programs process a C-tag of an IAS D-address by adding 600 to the IAS D-address at assembly time.

EXAMPLES:

Symbolic Input			*Assembled Output*		
ALO	9021C	1018	15	9621	1018
ALO	9021C	9036	15	9621	9036
ALO	9021C	8005	15	9621	8005

Tagged IAS I-Address. If the tag of the I-address of an instruction does not use the same indexing register as does the operation code, SOAP 4000 and SOAP 42 programs process an A-, B-, or C-tag of the IAS I-address by adding 200, 400, or 600 to the I-address at assembly time.

EXAMPLES:

Symbolic Input			*Assembled Output*		
RSL	2840	9017A	66	2840	9217
RSL	2840A	9017A	66	6840	9217
RSL	8003	9017B	66	8003	9417
AXB	0003	9017C	52	0003	9617

The 80xx D-addresses tagged by A or C are not accommodated by the SOAP 4000 or the SOAP 42 program. If such an instruction is encountered, the assembled output instruction will contain a blank D-address and the note, BLANK D, will be printed to the right of the assembled instruction.

As previously mentioned, an I-address tag must not be one that uses the same indexing register as does the operation code. If the SOAP 4000 or the SOAP 42 program encounters such an instruction (for example, AXA 1530 9012A), the output assembled instruction will contain a blank I-address and the note, BLANK I, will be printed to the right of the assembled instruction.

SPECIAL OPERATION

The following special operations of address modification are accomplished by SOAP 4000 and SOAP 42. By internal programming, these assembly programs produce two assembled output instructions for each input instruction that is tagged as indicated.

Tagged Drum I-Address. Instructions that include tagged drum I-addresses are processed as shown in the following examples.

Symbolic Input				*Symbolic Output*				*Assembled Output*			
XXXX	RAL	3103	0039A	XXXX	RAL	3103		XXXX	65	3103	YYYY
					NZA	0039A	0039	YYYY	40	4039	0039
XXXX	RAL	3103	0039B	XXXX	RAL	3103		XXXX	65	3103	YYYY
					NZB	0039B	0039	YYYY	42	0039	4039
XXXX	RAL	3103	0039C	XXXX	RAL	3103		XXXX	65	3103	YYYY
					NZC	0039C	0039	YYYY	48	4039	4039
XXXX	RAL	3103B	0039A	XXXX	RAL	3103B		XXXX	65	3103	(YYYY + 4000)
					NZA	0039A	0039	YYYY	40	4039	0039
XXXX	RAL	3103C	0039B	XXXX	RAL	3103C		XXXX	65	7103	(YYYY + 4000)
					NZB	0039B	0039	YYYY	42	0039	4039
XXXX	RAL	3103A	0039C	XXXX	RAL	3103A		XXXX	65	7103	YYYY
					NZC	0039C	0039	YYYY	48	4039	4039

Tagged IAS I-Address with B- or C-Tagged Drum D-Address. These instructions are processed as follows:

Symbolic Input				*Symbolic Output*				*Assembled Output*			
XXXX	RAL	3103C	9032A	XXXX	RAL	3103C		XXXX	65	7103	(YYYY + 4000)
					NZA	9032A	9032	YYYY	40	9232	9032
XXXX	RAL	3103B	9032C	XXXX	RAL	3103B		XXXX	65	3103	(YYYY + 4000)
					NZC	9032C	9032	YYYY	48	9632	9032

Tagged IAS I-Address with B-Tagged 80xx D-Address. These instructions are processed as follows:

Symbolic Input				*Symbolic Output*				*Assembled Output*			
XXXX	WTN	8010B	9032A	XXXX	WTN	8010B		XXXX	06	8010	(YYYY + 4000)
					NZA	9032A	9032	YYYY	40	9232	9032
XXXX	RAL	8005B	9032B	XXXX	RAL	8005B		XXXX	65	8005	(YYYY + 4000)
					NZB	9032B	9032	YYYY	42	9432	9032

5/cd Self-Loader Program

This routine can be used with the SOAP 4000 program. It loads itself and other programs that have been condensed in the five-instructions-per-card format. (Condensing a program can be done by assembling the symbolic program using the 5CD pseudo instruction.)

The 5/cd Self-Loader program deck is available only in symbolic format and consists of 52 single-instruction cards. It can be condensed, however, in five-per-card format by assembly, using SOAP 4000, resulting in ten condensed output cards. Word 1 of each condensed card of the 5/cd Self-Loader program has the format: 01 DDDD NNNN +. DDDD and NNNN represent, respectively, the deck or job number and card sequence number assigned by the user and punched in the 5CD card (see *5CD* under *Additional Pseudo Instructions).*

This program requires the first 47 consecutive drum locations in the region specified as Region W (xx01-xx47 or xx51-xx97) and seven other locations. Although Region W is specified by the first card of the 52-card deck supplied as being locations 1951-1997, it can be relocated, if desired, to any band as indicated previously.

Another feature of the 5/cd Self-Loader program is that card numbers of the condensed program being loaded are sequence-checked. A card-sequence error causes a programmed halt. If displayed, the program register indicates the contents of word 1 of the first card out of sequence. After correcting the card order, begin again, using a complete restart procedure. That is, remove the cards from the read feed and put the deck back in order. Then, starting with the first card of the 5/cd Self-Loader program, begin loading again.

Additional Pseudo Instructions

In addition to the pseudo-operation codes described in the *SOAP II Reference Manual*, the following seven pseudo instructions can be used as indicated. The first five, shown in Figure 32, are explained in *Programming Features of SOAP 2L and SOAP 2L Tape* of this bulletin.

5CD (FIVE INSTRUCTIONS PER CARD)

This pseudo instruction conditions the SOAP 4000 assembly program to produce the assembled output cards of an object program in five-instructions-per-card (condensed) format together with the usual single-instruction-per-card format. Figure 33 is an example of the 5CD pseudo instruction. The assembled output cards in five- and single-instruction-per-card formats are interspersed. They can be separated easily, however, by sorting, because columns 11-16 are blank in the one-per-card output but not blank in the five-per-card output. The instructions not condensed are pseudo instructions, type-1 (comments) and type-3 (blank) instructions, and instructions whose L-address is 800X. Literal constant cards, containing a 1 in column 41, *are* condensed. The card format of the five-per-card object program is shown in Figure 34.

Upon assembly, if the symbol table becomes packed (identified by the programmed halt 01 0111, and forcing partial assembly of the remaining cards), production of condensed output is suspended. It is resumed at the start of the multi-pass phase.

When a packed drum condition occurs (identified by the programmed halt 01 0222), further assembly resulting from pressing the program start key causes blank L-, D-, and I-fields (because no locations are available) in the single-instruction cards. *No blank fields* are punched, however, in the condensed cards.

Figure 33 shows the form of the 5CD pseudo instruction. The D-address entry can be either blank or a 4-digit number chosen by the user to identify the condensed object-program deck. This number is punched in columns 3-6 of each condensed card. If the D-address of the 5CD pseudo instruction is blank, 0000 is punched in columns 3-6 of each condensed card. The I-address entry of the 5CD pseudo instruction can be either blank or a 4-digit number, used to specify the first (starting) sequence-card number of the five-per-card deck. A card-sequence number is punched in columns 7-10 of each condensed object-program card, starting with either the number specified in the I-address of the 5CD pseudo instruction, or 0001 if the I-address of the 5CD pseudo instruction is blank.

One application of the sequence number that is automatically punched in each condensed object-program card is to use the sequence-checking feature of a loading program (see *5/cd Self-Loader Program*).

Pseudo Operation	SOAP 4000	SOAP 42
DLA		
As Input	yes	yes
As Output	no	yes
PST	yes	yes
RDR	yes	yes
SEQ	yes	yes
TAP	yes	no

Figure 32. Five Additional Pseudo Instructions

41	42	43	44- -47	48	49. 50	51	52- -55		56	57- -60		61- -72	S.O.A.P. I COL.
41	42	43	44- -47	48	49,50	51	52- -55	56	57	58- -61	62	63- -72	S.O.A.P. II COL.
T P	S N	LOCATION		OPER. CODE		DATA ADDRESS		T A G	INSTR. ADDRESS		T A G	REMARKS	ACCU UPPER 8003
				5	C D	1 2 3 4			0 0 1 1				

Figure 33. 5CD Pseudo Instruction

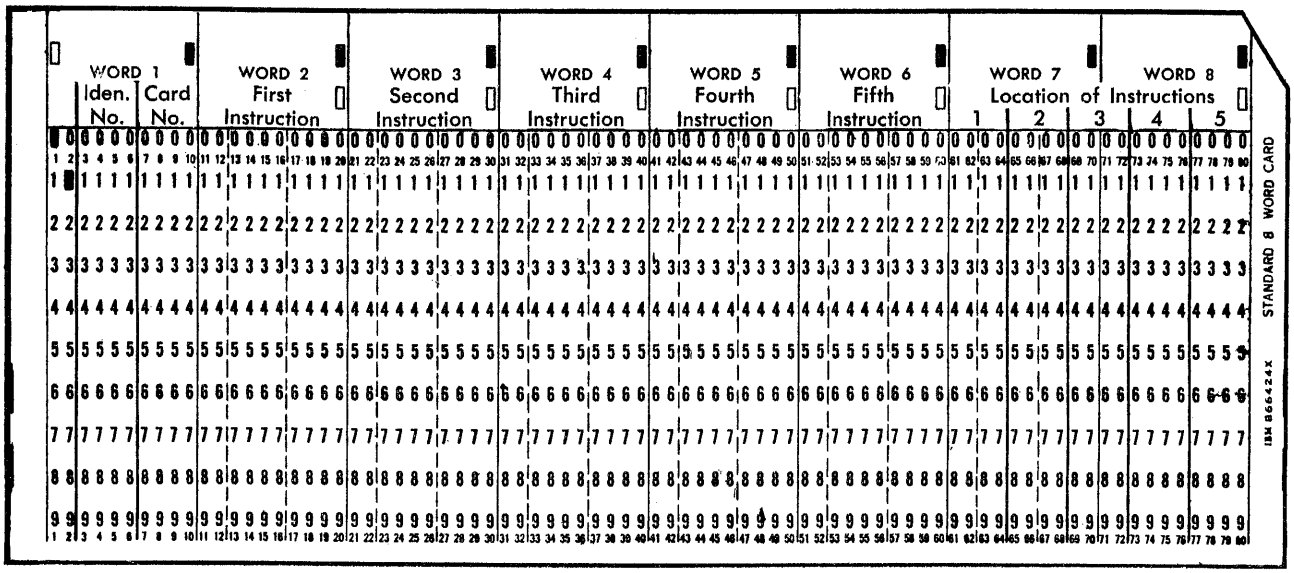


Figure 34. Format of Five-Instructions-Per-Card Assembled Output

General Description of Producing Assembled Output. Briefly, assembled output is produced as follows:

- The first instruction (or constant, as the case may be) is stored in the one-per-card punch area.
- The single-instruction assembled output card is punched.
- The first instruction is tested to determine if it is condensable. If it is, the instruction is stored in word 2 of the condensed punch area. The assigned L-address is stored in the four high-order positions of word 7 of the condensed punch area. If not condensable, the first instruction is not stored in the condensed punch area.

This process is repeated, producing the single-instruction assembled output cards and storing only the condensable instructions in the condensed punch area. This occurs until the fifth condensable instruction is stored in word 6 of the condensed punch area and its assigned L-address is stored in the last four (low-order) positions of word 8. At this time a condensed output card is punched, in the format just described and as shown in Figure 34. Thus the one-per-card and the five-per-card output of the symbolic assembly program is produced, except for punching out the last *condensed* card. Punching this card usually requires further instruction, as explained in the following paragraph.

As mentioned, a condensed card is punched after the fifth instruction is stored in the condensed punch area. On the average, one program in five contains condensable instructions that are in multiples of five. In this one case in five, the last condensed card is produced in the same manner as the condensed cards preceding it. In the remaining four cases, however, punching of the last condensed card awaits the signal that the fifth instruction to be condensed is stored. Causing this last condensed output card to be punched is a function of the XFR pseudo instruction, explained next.

XFR (TRANSFER)

The pseudo instruction, XFR XXXX (blank), can be used in a symbolic source program that is to be assembled by SOAP 4000. The instruction produces a transfer card. The transfer card *produced* becomes part of the assembled object program and is loaded with it. After the assembled object program is loaded, program operation begins automatically, because the transfer card switches program control to the location of the first object-program instruction (starting address). The XFR pseudo-instruction card must be placed in the symbolic source-program deck just following the last program-instruction card. (This is usually just ahead of the PST or PAT pseudo-instruction card.) The entry in the D-address part of the XFR pseudo instruction is the starting address, which may be in absolute, regional, or symbolic form. If the D-address contains a previously-undefined symbol, a location is assigned to the symbol and included in the (output) transfer card.

The XFR pseudo instruction serves an additional purpose. It causes the last (condensed) output card to be punched in cases where there are one, two, three, or four instructions remaining to be condensed after the next-to-last condensed card is punched.

41	42	43	44- -47	48	49,50	51	52- -55		56	57- -60		61- -72	SOAP I COL.
41	42	43	44- -47	48	49,50	51	52- -55	56	57	58- -61	62	63- -72	SOAP II COL.
T P	S N	LOCATION		OPER. CODE		DATA ADDRESS		TAG	INSTR. ADDRESS		TAG	REMARKS	ACCU UPPER 8003
1		E	X A M P	L	E	2 b C A L		C	U	L A T E		G R O S S P A Y	
1													
				5	C D	0 0 0 2				0 0 1 1			
				B	L R	1 9 5 1				1 9 6 0		R E A D A R E A	
				R	E G	P 0 0 2 7				0 0 3 0		P U N C H A R E A	
				B	L R	0 0 0 0				0 0 0 9		R A T E T A B L E	
				S	Y N	C G				0 2 0 0			
1													
		S	T A R T	R	C D	1 9 5 0						R E A D C A R D	
				L	D D	1 9 5 1						S T O R E	
				S	T D	P 0 0 0 1						I D E N T	
				R	A L	1 9 5 2						G E T H O U R L Y	
				S	T D	P 0 0 0 2						R A T E	
				A	L O	M S K				8 0 0 2		6 0 0 0 0 0 0 2 0 0	
		C	G	M	P Y	1 9 5 3						C A L C U L A T E	
				S	T D	P 0 0 0 3						G R O S S	
				S	R D	0 0 0 1						R O U N D	
				S	T L	P 0 0 0 4							
				P	C H	P 0 0 0 1			S	T A R T		P U N C H	
				X	F R	S T A R T							

Figure 35. Sample Program Using a Literal Constant and the Pseudo Operations, 5CD and XFR

An XFR pseudo instruction with a blank D-address, used in a source program that includes a 5CD card, does not cause a transfer card to be punched out. It does cause the last condensed card to be punched, however, as previously explained.

Figure 35 is a minor modification of the program shown in *Part I, Figure 4* of the *SOAP II Reference Manual.* This short program, Calculate Gross Pay, illustrates the use of the pseudo-operations, 5CD and XFR, as well as a literal constant. The program is identified as job number 0002. The assembled program is to be condensed using a 5CD card. A ten-card, self-loading, loading program is available to load the (condensed) assembled program. Therefore, the condensed cards are sequence-numbered beginning with the number 0011. A transfer card is produced with the assembled output, which causes program control to be switched to the location assigned to the symbol START (i.e., 0050) after the assembled output is loaded. Figure 36 is a listing of assembled output.

When loading the condensed program deck and running the program, set the storage-entry switches at 70 1952 9999. After the condensed program deck (including output transfer card) is loaded, program execution begins automatically if the user supplies the input data cards.

Operating Instructions

Figures 37 and 38, respectively, are the operating instructions for SOAP 4000 and SOAP 42.

Single-Instruction-Per-Card Listing (together with an indication of the order of the condensed cards):

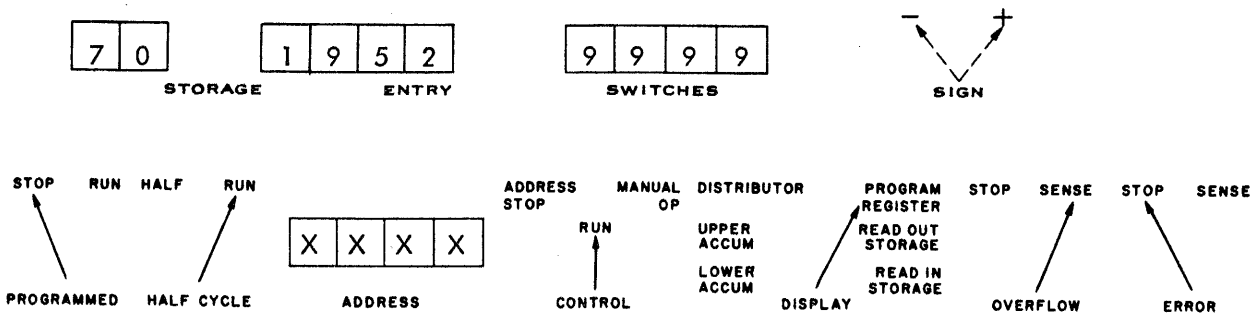
```
 1   1                    EXAMPLE 2 CALCULATE GROSS PAY
 2   1
 3   0              5CD     0002     0011
 4   0              BLR     1951     1960       READ AREA
 5   0              REG    P0027     0030       PUNCH AREA
 6   0              BLR     0000     0009       RATE TABLE
 7   0              SYN    CG        0200
 8   1
 9   0    START     RCD     1950                READ CARD      0050    70  1950  0100
10   0              LDD     1951                STORE          0100    69  1951  0054
11   0              STD    P0001                IDENT          0054    24  0027  0080
12   0              RAL     1952                GET HOURLY     0080    65  1952  0057
13   0              STD    P0002                RATE           0057    24  0028  0031
     (First Condensed Card - See Separate Listing Below)
14   0              ALO      MSK     8002       6000000200     0031    15  0034  8002
15   1                       LIT                6000000200     0034    60  0000  0200
16   0    CG        MPY     1953                CALCULATE      0200    19  1953  0023
17   0              STD    P0003                GROSS          0023    24  0029  0032
18   0              SRD     0001                ROUND          0032    31  0001  0039
     (Second Condensed Card - See Separate Listing Below)
19   0              STL    P0004                               0039    20  0030  0033
20   0              PCH    P0001    START       PUNCH          0033    71  0027  0050
21   0              XFR    START
     (Third Condensed Card - See Separate Listing Below)
     (Last Condensed Card - Output Transfer Card - See Separate Listing Below)
```

Five-Instructions-Per-Card Listing:

```
0100020011& 70195001006& 69195100546& 24002700806& F5195200576& 24002800316& 00500100006& 54008000576&
0100020012& 15003480026& 60000002006& 19195300236& B4002900326& 31000100396& 00310034026& 00002300326&
0100020013& 20003000336& 71002705006& 19195300236& B4002900326& 31000100396& 00390033026& 00002300326&
0100020014- 00000000506& 00000000006& 00000000006& 00000000006& 00000000006& 00000000006& 00000000006&
```

Figure 36. Assembled Output Listing

IBM 650 PROGRAM OPERATING INSTRUCTIONS



Initial Console Setting as shown above.

A. Normal Starting Procedure: Computer Reset; Program Start.

B. Special Instructions

If SOAP 4000 is already on the drum do one of the following:

1. Set 00 0000 1000 in the Storage Entry switches.
2. Precede input with a B O P card and set 00 0000 1950 in the Storage Entry switches.

Set Sign switch to minus (-) if automatic symbol-table punch-out is desired.

Card Input - Output

READ FEED

NO. OF CARDS	FILE DESCRIPTION
367	SOAP 4000 (including loader)
xxx	Symbolic program deck
1	PST card (if desired)

PUNCH FEED

CARD FORM
SOAP II

CONTROL PANELS

SOAP 2L for the appropriate
card Input-Output
components

TAPE UNITS

ADDRESS	INPUT, OUTPUT OR OTHER	FILE PROTECTION RING IN	FILE PROTECTION RING OUT	LABEL CHARACTERISTICS	FILE DESCRIPTION
8010					
8011					
8012	Library Tape		X	(Optional—used only if TAP pseudo instruction is used)	
8013					
8014					
8015					

Figure 37 (1). SOAP 4000 Operating Instructions

Other Instructions and Remarks

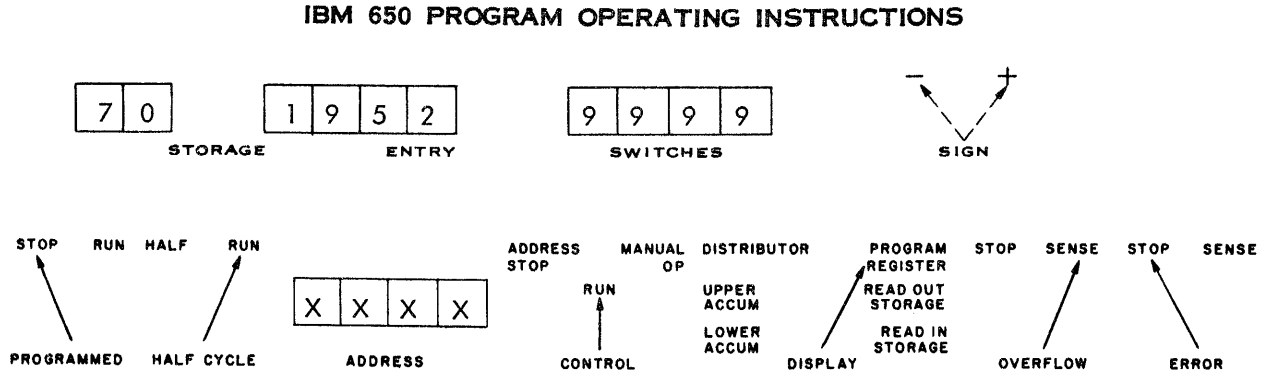
After each machine pass, discard the last card out of the punch feed. If the preceding card contains a 9-punch in column 80, or if it is an availability-table card, assembly is complete. If the card does not meet either of these conditions remove from the deck all cards that contain a 9-punch in column 80. Use the remainder of the deck for input to the next pass. The first card that does not contain a 9-punch in column 80 (i.e., the first card of the input to the next pass) will contain a 4-punch in that column.

An availability table punch-out may be initiated manually by transferring to location 1900, and a symbol-table punch-out by transferring to location 1800.

Program Stops and Required Action

STOP ADDRESS	MESSAGE - EXPLANATION - ACTION
0222	No locations available for the remaining portion of the program being assembled. Press the Program-Start key to continue assembly. Addresses not assigned will be left blank in the output cards.

Figure 37 (2). SOAP 4000 Operating Instructions



Initial Console Setting as shown above.

A. Normal Starting Procedure: Computer Reset; Program Start.

B. Special Instructions:

If SOAP 42 is already on the drum do one of the following:

1. Set 00 0000 1000 in the Storage Entry switches.
2. Precede input with a B O P card and set 00 0000 1950 in the Storage Entry switches.

Set Sign switch to minus (-) if automatic symbol-table punch-out is desired.

Figure 38 (1). SOAP 42 Operating Instructions

Card Input - Output

READ FEED

NO. OF CARDS	FILE DESCRIPTION
297	SOAP 42 (including loader)
xxx	Symbolic program deck
1	PST card (if desired)

PUNCH FEED

CARD FORM
SOAP II

CONTROL PANELS

SOAP 2L for the appropriate
card Input-Output
components

TAPE UNITS

ADDRESS	INPUT, OUTPUT OR OTHER	FILE PROTECTION RING		LABEL CHARACTERISTICS	FILE DESCRIPTION
		IN	OUT		
8010					
8011					
8012					
8013					
8014					
8015					

Other Instructions and Remarks

After each machine pass, discard the last card out of the punch feed. If the preceding card contains a 9-punch in column 80, or if it is an availability-table card, assembly is complete. If the card does not meet either of these conditions remove from the deck all cards that contain a 9-punch in column 80. Use the remainder of the deck for input to the next pass. The first card that does not contain a 9-punch in column 80 (i.e., the first card of the input to the next pass) will contain a 4-punch in that column.

An availability table punch-out may be initiated manually by transferring to location 1900, and a symbol-table punch-out by transferring to location 1800.

Program Stops and Required Action

STOP ADDRESS	MESSAGE - EXPLANATION - ACTION
0222	No locations available for the remaining portion of the program being assembled. Press the Program-Start key to continue assembly. Addresses not assigned will be left blank in the output cards.

Figure 38 (2). SOAP 42 Operating Instructions

IBM®
International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, New York

Printed in U.S.A. J24-5013-0