304 *Neat* MANUAL

National's Electronic Autocoding Technique FOR THE NCR 304 ELECTRONIC DATA PROCESSING SYSTEM

National

304 *Neat* MANUAL

National's Electronic Autocoding Technique FOR THE NCR 304 ELECTRONIC DATA PROCESSING SYSTEM

National *

X-1742-10 11-1-58 The NCR Co.

Neat

The Neat System consists of:

- The Neat ***FORMAT,*** in which the National 304 Electronic Data Processor is programmed, and
- The Neat ***ASSEMBLY*** program, which automatically translates the programmer's code from Neat ***FORMAT*** into final Processor code.

The Neat ***FORMAT*** furnishes the programmer with:

- A simple format for each Instruction;
- An easily-remembered abbreviation for the name of each operation;
- Reference to data-fields by name, rather than by Address and Partial-Word;
- Direct naming of Constants within each Instruction;
- Designation of the sequence of Instructions by Reference Numbers (sometimes called "Relative Addresses" or "Floating Addresses") rather than by fixed Memory addresses, freely permitting insertion and deletion of Instructions in the program;
- Automatic inclusion of STEP (Standard Tape Executive Program) for all File housekeeping operations.

Since the individual Processor Instructions in the NCR 304 System are on generally the same level of thought as the "pseudo-instructions" used in conventional compiling systems, Neat furnishes all the freedom and convenience of such compilers, yet the programmer always writes the actual Processor Instructions which will be executed. Optimum programming is therefore easy to achieve, and code-checking is performed rapidly and conveniently.

This Manual discusses the use of Neat ***FORMAT*** by the programmer in writing his programs; operation of the Neat ***ASSEMBLY*** program is described in a separate Neat Operating Manual. Our present discussion assumes a complete familiarity with the NCR 304 Programming Manual, and is conducted in tutorial form. The details are then described in systematic fashion for reference purposes in a series of Appendices which appear at the back of this Manual:

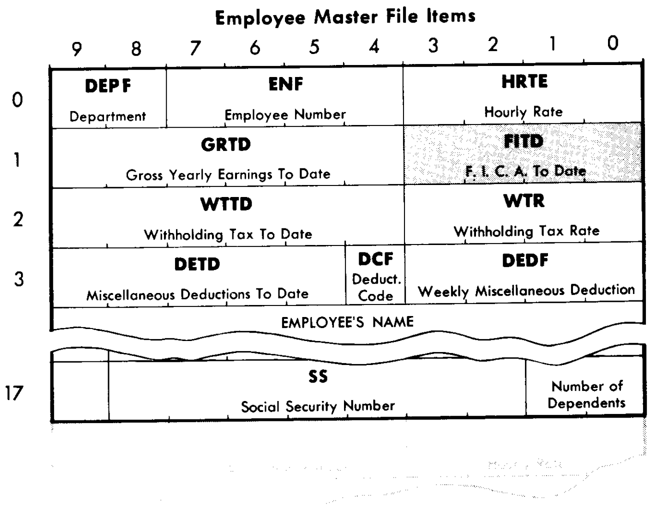
APPENDIX A:	Neat Sheets
	Header Sheet
	File Specification Sheet
	Data Defining Sheet
	Coding Sheet
APPENDIX B:	STEP (Standard Tape Executive Program)
APPENDIX C:	Neat Instruction Formats

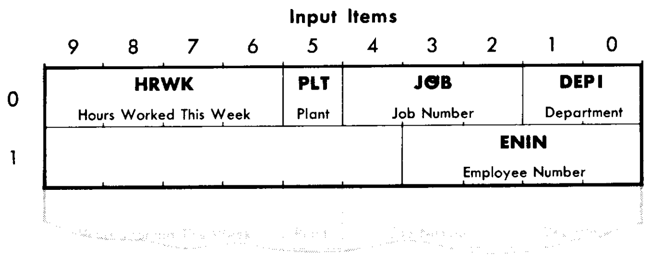
The material in this Manual is designed to teach the programmer to write programs in Neat ***FORMAT*** and to give him a general understanding of the functions performed by the Neat ***ASSEMBLY*** program. It is supplemented by the Neat Operating Manual which provides keypunching instructions for paper tape and punched cards, detailed operating instructions for using the Neat ***ASSEMBLY*** showing all available options and all possible printouts, and a more complete description of the Neat ***ASSEMBLY*** itself. The Neat

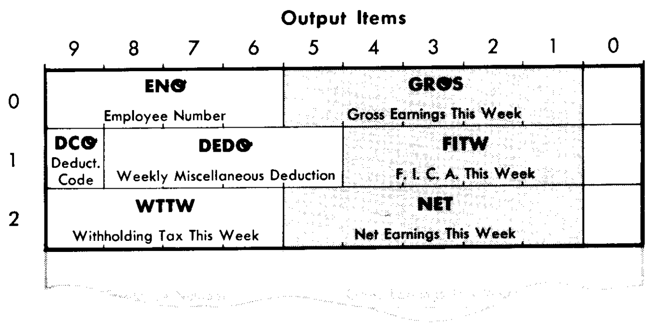
Operating Manual also discusses the Librarian Program which places each assembled program on the Program-Library Tape.

EXAMPLE 1:

As an initial example of coding in *Neat* ***FORMAT*** consider a small portion of the program for processing an industrial payroll. Earlier portions of the program have brought into Memory a string of File Items and a string of Input Items, and are in process of building up a string of Output Items. The format of each of these items is:

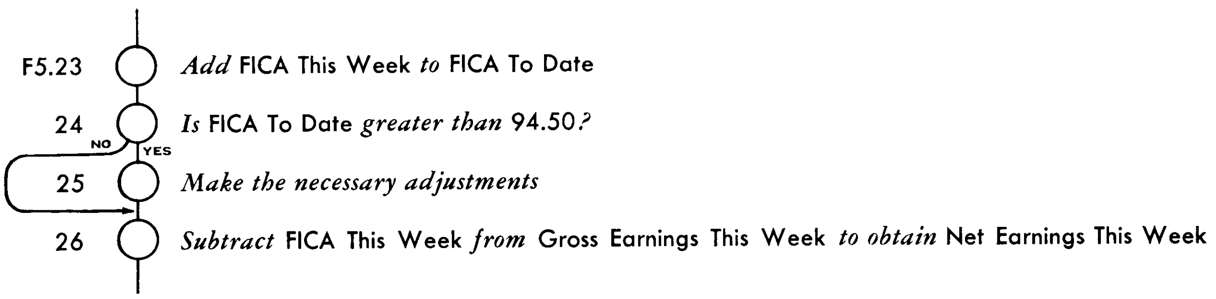

Employee Master File Items
9 8 7 6 5 4 3 2 1 0
0
DEPF Department
ENF Employee Number
HRTE Hourly Rate
1
GRTD Gross Yearly Earnings To Date
FITD F. I. C. A. To Date
2
WTTD Withholding Tax To Date
WTR Withholding Tax Rate
3
DETD Miscellaneous Deductions To Date
DCF Deduct. Code
DEDF Weekly Miscellaneous Deduction
EMPLOYEE'S NAME
17
SS Social Security Number
Number of Dependents




Output Items
9 8 7 6 5 4 3 2 1 0
0
ENØ Employee Number
GRØS Gross Earnings This Week
1
DCØ Deduct. Code
DEDØ Weekly Miscellaneous Deduction
FITW F. I. C. A. This Week
2
WTTW Withholding Tax This Week
NET Net Earnings This Week

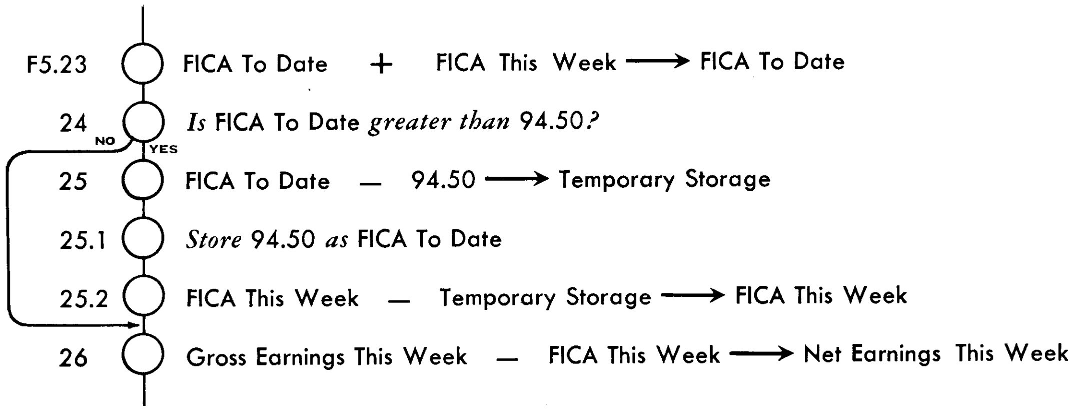
A convenient, but arbitrary, section of the overall program has been designated "Region F5" and within that Region **Gross Earnings This Week** and **FICA This Week** have already been computed and stored in the Output Item. In all, 22 steps have been written within this Region and, starting with step 23, we wish to accumulate **FICA To Date** and **Net Earnings This Week**, subject to the condition that **FICA To Date** may not exceed $94.50.

The initial flow chart for this portion of the program will appear as:

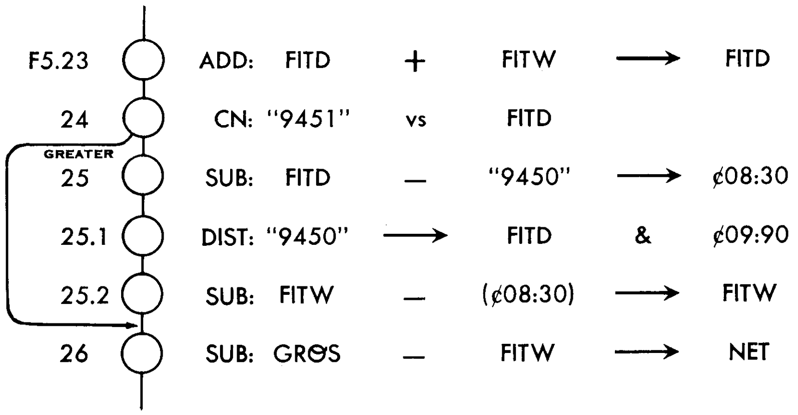

F5.23 *Add* FICA This Week *to* FICA To Date
24 *Is* FICA To Date *greater than* 94.50?
NO
YES
25 *Make the necessary adjustments*
26 *Subtract* FICA This Week *from* Gross Earnings This Week *to obtain* Net Earnings This Week

At the time this flow chart is written it will be realized that step 25 will actually require several Instructions to accomplish, but study of such a detail is not appropriate at this stage of developing the program.

After the logical structure of the entire program has been established, another flow chart will be drawn, showing all the detail omitted previously, and essentially containing one step for each Instruction:


F5.23 FICA To Date + FICA This Week → FICA To Date
24 Is FICA To Date greater than 94.50?
NO
YES
25 FICA To Date — 94.50 → Temporary Storage
25.1 Store 94.50 as FICA To Date
25.2 FICA This Week — Temporary Storage → FICA This Week
26 Gross Earnings This Week — FICA This Week → Net Earnings This Week

The more precise re-drawing of this flow chart will name the actual Instructions to be used, and will refer to the data fields by their specific designators:


F5.23 ADD: FITD + FITW → FITD
24 CN: "9451" vs FITD
GREATER
25 SUB: FITD — "9450" → ¢08:30
25.1 DIST: "9450" → FITD & ¢09:90
25.2 SUB: FITW — (¢08:30) → FITW
26 SUB: GRØS — FITW → NET

The final *Neat* code for this portion of the program will be merely a transcription of this last flow chart to a columnar sheet:

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &	
Region	Position													
F5	230		ADD				FITD		FITW		FITD			>
	240		CN				9451	C	FITD		F5260			>
	250		SUB				FITD		9450	C	¢0830			>
	251		DIST				9450	C	FITD		¢0990			>
	252		SUB				FITW		¢0830		FITW			>
	260		SUB				GRØS		FITW		NET			>
														>

Note that Reference Number F5.26.0 is specified as the Jump Address for the COMPARE NUMERIC Instruction (step 24).

Note also the use of the Tag "C" to indicate that "9451" and "9450" are Constants; a Constant named within an Instruction in this fashion is called a Named Constant. The *Neat* ***ASSEMBLY*** places all Named Constants into the irrelevant fields of Instructions in the program, and then cross-indexes the stored locations of these Constants into the Instructions which name them.

Note also the means of referring to the "Special" Cells. When programming in *Neat* ***FORMAT*** it is sometimes necessary to refer to an actual Memory address; such an address may be either one of the "Special" Cells, or a Cell in Main Memory (in the latter case, almost invariably one of the Index Registers).

The "Special" Cells are referred to by means of the normal 3-character Address and the Partial-Word, as illustrated for ¢08:30 (steps 25 and 25.2) and for ¢09:90 (step 25.1).

A Cell in Main Memory is referred to by its 4-digit address (there are no Address-Type Numbers in *Neat* **FORMAT**) and Partial-Word, using an auxiliary line on the Coding Sheet as illustrated in Example 2, below.

In order that the *Neat* **ASSEMBLY** may translate the Field Designators into Memory addresses, the programmer makes up a Data Defining Sheet for each kind of Item referred to by the program. On this sheet he lists the Designator for each Field in the Item, and shows its location within the Item. The program which has been written in *Neat* **FORMAT**, and the information on the Data Defining Sheets, are punched into paper tape or cards, and they are read into the Processor by the *Neat* **ASSEMBLY** program. The information on the Data Defining Sheets becomes a "dictionary" by which the *Neat* **ASSEMBLY** can translate each Field Designator into an actual Memory address and Partial-Word, as part of the process of assembling the code, written in *Neat* **FORMAT**, into a finished program.

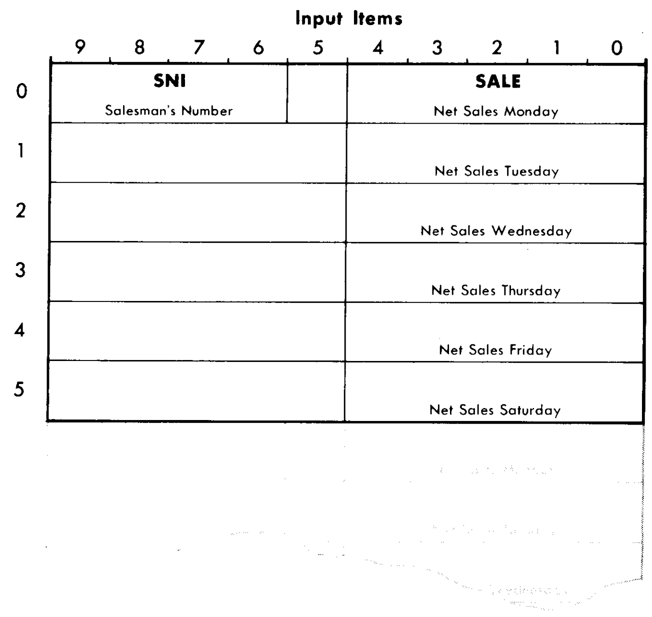
Field Designators, then, are used in *Neat* ***FORMAT*** as convenient references to the locations of data-fields within an Item. Even though the same information may appear in different fields, each field must be assigned a unique Designator. Thus in the previous example we have Employee Number (Input) **ENIN**, Employee Number (File) **ENF**, and Employee Number (Output) **ENØ**.

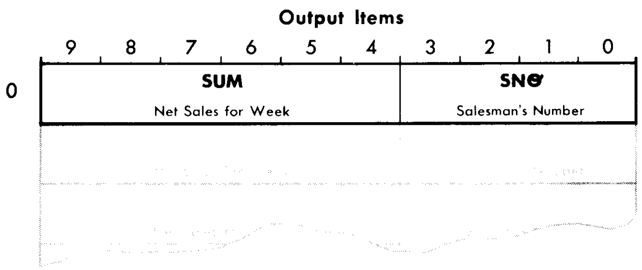
Many programmers prefer to use mnemonic, or suggestive, abbreviations as Field Designators, as in the previous example. Others feel that a mnemonic system becomes so elaborate in actual use that it is no longer suggestive in any practical sense; they prefer to use an arbitrary alphanumeric code to designate fields, and have the programmer look at the Item layouts each time he uses a particular field in his program, rather than rely on his memory for the designators.

This Manual takes no position in this controversy, and will make no attempt to recommend one method over the other. Mnemonic Field Designators are used in the examples because they appear to be more convenient for training purposes, and their use here does not necessarily imply any recommendation of their use in production programming.

EXAMPLE 2:

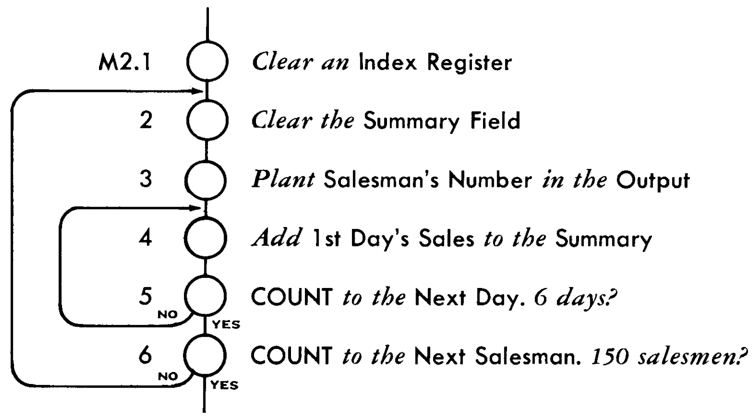
The next example introduces the use of an Index Register, and shows the programming to summarize the weekly sales of each of 150 salesmen. For purpose of this example, the SUMMARIZE Instruction is not used.



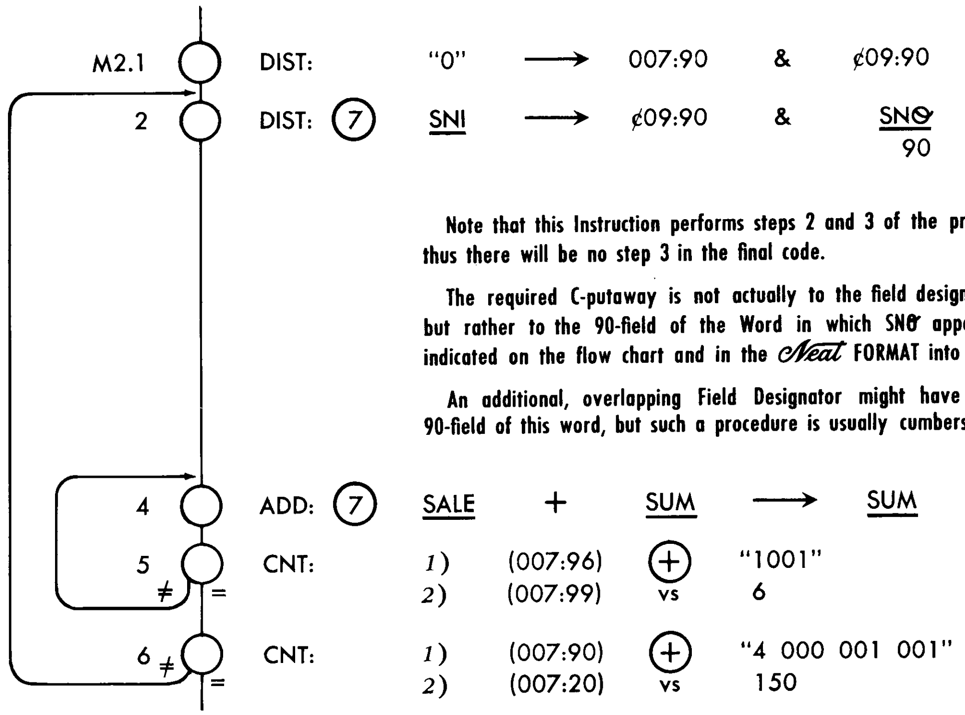


Blank areas in the Input Items represent additional information which would be stored in the Item, but which is not pertinent to this example.

The initial flow chart will be:


M2.1
2
3
4
5
6
NO
YES
NO
YES
Clear an Index Register
Clear the Summary Field
Plant Salesman's Number *in the* Output
Add 1st Day's Sales *to the* Summary
COUNT *to the* Next Day. *6 days?*
COUNT *to the* Next Salesman. *150 salesmen?*

In the final flow chart, the symbol ⑦ which appears in Instructions 2 and 4 indicates that each of these Instructions uses Index Register 7. Underlining individual syllables within an Instruction indicates that those syllables are to be relative to the Index Register.


M2.1 DIST: "0" ⟶ 007:90 & ¢09:90
2 DIST: ⑦ SNI ⟶ ¢09:90 & SNØ 90
4 ADD: ⑦ SALE + SUM ⟶ SUM
5 CNT: 1) (007:96) ⊕ "1001" 2) (007:99) vs 6
≠ =
6 ≠ CNT: 1) (007:90) ⊕ "4 000 001 001" 2) (007:20) vs 150
≠ =

Note that this Instruction performs steps 2 and 3 of the previous flow chart, and thus there will be no step 3 in the final code.

The required C-putaway is not actually to the field designated SNØ (the 30-field) but rather to the 90-field of the Word in which SNØ appears. Note how this is indicated on the flow chart and in the *Neat* FORMAT into which it is transcribed.

An additional, overlapping Field Designator might have been defined for the 90-field of this word, but such a procedure is usually cumbersome and inconvenient.

In transcribing this flow chart into *Neat* **FORMAT**, it will be promptly observed that the 10-digit Constant "4 000 001 001" required by Instruction 6 cannot be written in a 5-place column on the Coding Sheet. Any Constant of more than 5 characters is placed by the programmer into a separate word of Constants (See CNST, Appendix page C-10) and the Instruction refers to it by "Address" (ie- Reference Number) and Partial-Word. Often all long Constants used anywhere in the program will be placed in a single Region; the example shows this Constant placed in Position 1 of Region CN with, therefore, Reference Number CN.01.0.

Reference No.			M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&	
Region	Position														
M2	01	0		DIST				0	C	0007		¢0990			>
										90					>
	02	0		DIST			7	SNI	X	¢0090		SNØ	X		>
												90			>
	04	0		ADD			7	SALE	X	SUM	X	SUM	X		>
	05	0		CNT		769		1001	C	006		M2040			>
	06	0		CNT		70C		CN010		150		M2020			>
								90							>
															>

Reference No.			M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&	
Region	Position														
CN	01	0		CNST		4		000	C	001	C	001	C		>
															>

Note the use of the Tag "X" to designate those syllables of an Instruction which are relative to the Index Register.

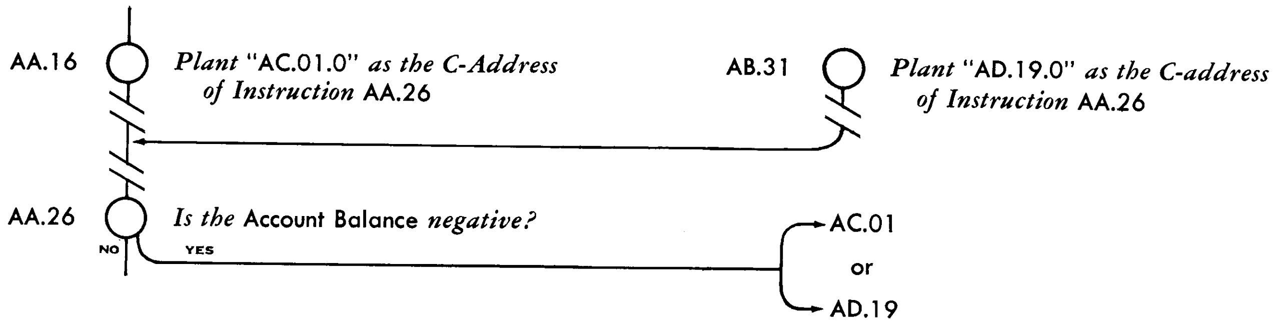
The format of CNT is explained in the Appendix, page C-5. The B-syllable of CNT is not tagged "C", since 006 is not a Constant *used* by the Instruction; it is part of the Instruction itself.

In this example, there is no step 3 in the program. Since the Reference Numbers indicate only the *relative* positions of the Instructions within the program sequence, the *Neat* ***ASSEMBLY*** will cause step 4 to follow immediately after step 2 in the final program. Similarly, in Example 1 we were able to interpolate additional Instructions (Steps 25.1 and 25.2) *into* the program without difficulty.

Remember | The program must always contain the Instructions necessary to accomplish all initial presetting of Index Registers.

EXAMPLE 3:

There are occasions when it is necessary to treat a Reference Number as though it were a Constant, as in the following illustration:


AA.16
Plant "AC.01.0" *as the C-Address of Instruction* AA.26
AB.31
Plant "AD.19.0" *as the C-address of Instruction* AA.26
AA.26
Is the Account Balance *negative?*
NO
YES
AC.01
or
AD.19

In writing Instruction AA.16 the programmer must not use Tag "C" with the A-column, as that would cause the *Neat* ***ASSEMBLY*** to treat the contents of that column as a Named Constant, and store the characters "AC010" as a 5-character field. What the programmer actually wants the *Neat* ***ASSEMBLY*** to store as a Constant is the 3-character Address-Type field containing the actual Memory address into which Reference Number AC.01.0 will be translated. "AC010" is, therefore, not *yet* a Constant, but it is rather a Constant "once removed"; the *Neat* ***ASSEMBLY*** must:

a) Translate the Reference Number into an actual Memory address;

b) *Then* treat the resulting Memory address as though it were a Named Constant.

A Constant "once removed" is specified by using the Tag "R". Thus Instruction AA.16 would be written:

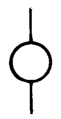
Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&	
Region	Position													
AA	160		DIST				AC010	R	¢0990		AA260			>
											20			>

Note that, in the C-column, Reference Number AA.26.0 will be translated by the Neat ***ASSEMBLY*** into the Memory address of the first word of Instruction AA.26; therefore this Reference Number is entered as the address of the putaway, while the Partial-Word field of the putaway is specified in the auxiliary line.

Field Designators, Item Designators (not yet defined), 4-digit Memory addresses, and 4-digit Tallies, when desired as Constants, are all specified as Constants "once removed", since they each must *first* be translated into an Address or an Address-Type Number, and *then* treated as a Named Constant.

EXAMPLE 4:

Suppose now that we are required to operate on the *second* word of an Instruction:

R3.02  *Plant* "86" *as the* AL AR *of Instruction* S4.11

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&	
Region	Position													
R3	020		DIST				86	C	S4110		¢0990			>
									154					>

The Reference Number S4.11.0 specified for the B-putaway will initially be translated by the Neat ***ASSEMBLY*** into the Memory address of the first word of Instruction S4.11; but now the auxiliary line specifies the increment **1** which is to be added to the specified address after translation, as well as the Partial-Word **54** within the desired word. Therefore, the B-putaway will be made to the 54-field of the *second* word of Instruction S4.11.

STORAGE OF NAMED CONSTANTS:

After the code, written in Neat **FORMAT**, has been key-punched and then read into the Processor by the Neat ***ASSEMBLY*** program, each column on the Coding Sheet (between the double-ruled lines) is placed in Memory right-justified in a separate word.* (It will be seen that the practice of writing left-zeros on the Coding Sheet, as shown in the examples up to now, is optional at the programmer's taste.) Whenever a column contains a Named Constant (that is, carries Tag "C" within an Instruction) the Neat ***ASSEMBLY*** counts the number of significant characters in it, and adds it to its own list of Constants. Then, after all duplications have been eliminated, the Constants are placed into the irrelevant fields of Instructions in the program. If a program should name more Constants than can be fitted into the available fields, the Neat ***ASSEMBLY*** will set up additional words of Constants at the end of the program. After all Named Constants have been stored, their locations will be cross-indexed into the Instructions which name them.

*Complete instructions for key-punching Neat **FORMAT** into paper tape or cards, including the extent to which blank columns must be punched as zeros, are given in the Neat Operating Manual.

However, if the programmer should wish to name the 3-character Constant "002" for example, the Neat **ASSEMBLY** could not distinguish it from "2", and would store it in a 1-character field. Therefore the programmer requires some means of reserving significant left-zeros in Named Constants.

RESERVING LEFT-ZEROS IN NAMED CONSTANTS:

The programmer may unambiguously define the field length of a Named Constant by substituting the character "+" for the leftmost significant zero. Whenever the Neat **ASSEMBLY** finds a "+" in that position, it will change the "+" into a "0" but will also reserve the full-size field for the Constant. This translation of "+" into "0" occurs only in the leftmost significant character-position, so that:

NAMED CONSTANT	WILL BE STORED AS
+02	002
++05	0+05
B++05	B++05

This technique raises the question of naming a Constant such as "+&d" which might be required to establish some specific binary configuration. Such a Constant may be entered in either of two ways:

a) Name the Constant, in the Instruction, as "++&d", in which case it will be stored as a 4-character field "0+&d" which will serve just as well for right-justified operations;

b) Place the Constant, as "+&d", into a Pseudo-Instruction CNST (Appendix page C-10) where the problem of reserving left-zeros does not arise.

The method of writing a "+" to mark the leftmost significant zero is also used when the programmer specifies, in an auxiliary line to an Instruction, that the 00-field of a word is addressed; he writes it as "+00". Suppose that in Example 1, page 2, the program included a test to see whether the Department Number (Input) is an odd number. This would appear on the Coding Sheet as:

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&	
Region	Position													
F5	040		TBIT				DEPI		1	C	G7160			>
							+00							>

SINGLE-LINE REFERENCE TO THE INDEX REGISTERS:

The technique of using the character "+" to reserve left-zeros also allows the programmer to address any of the Index Registers by designating the Address and Partial-Word within 5 characters on a single line of the Coding Sheet, just like references to the Special Cells. For this purpose the first 10 Cells of Memory are given the pseudo-addresses +00 through +09, and the first Instruction in Example 2, page 6, could have been written:

Reference No.														
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &	
M2	010		DIST				0	C	+0790		¢0990			>
								X	¢0090			X		>

There can be no ambiguity between this reference to Cell 007, and a reference to the Cell whose actual Memory address is +07, because *there are no Address-Type Numbers* in Neat **FORMAT**, and the latter Cell may only be referred to by the 4-digit address 2007; in fact, the programmer will always think of it as Cell 2007, not as Cell +07.

OTHER COLUMNS ON THE CODING SHEET:

We have covered, informally, most of the columns on the Neat Coding Sheet. All possible entries are tabulated systematically in Appendix A, but for completeness the remaining columns are briefly described here also:

M:	Automonitor Digit.
V:	Variation Designator. However, observe that the variations defined in Neat **FORMAT** do not exactly correspond to those in the final code. The Neat **FORMAT** Operations and their Variations have been defined for the maximum convenience to the programmer.
Misc:	The information entered in this column is specifically defined for each Instruction which uses it. See CNT in example 2, page 6, for illustration.
SP&:	S in this column indicates that the Sign of V is to be negative. P in this column Protects the Instruction against the storage of any Named Constants within it. Sometimes a complete Instruction will be replaced by the program itself, and in that case it is essential that no Constants be stored within that Instruction. & in this column indicates *both* minus-V *and* Protect.

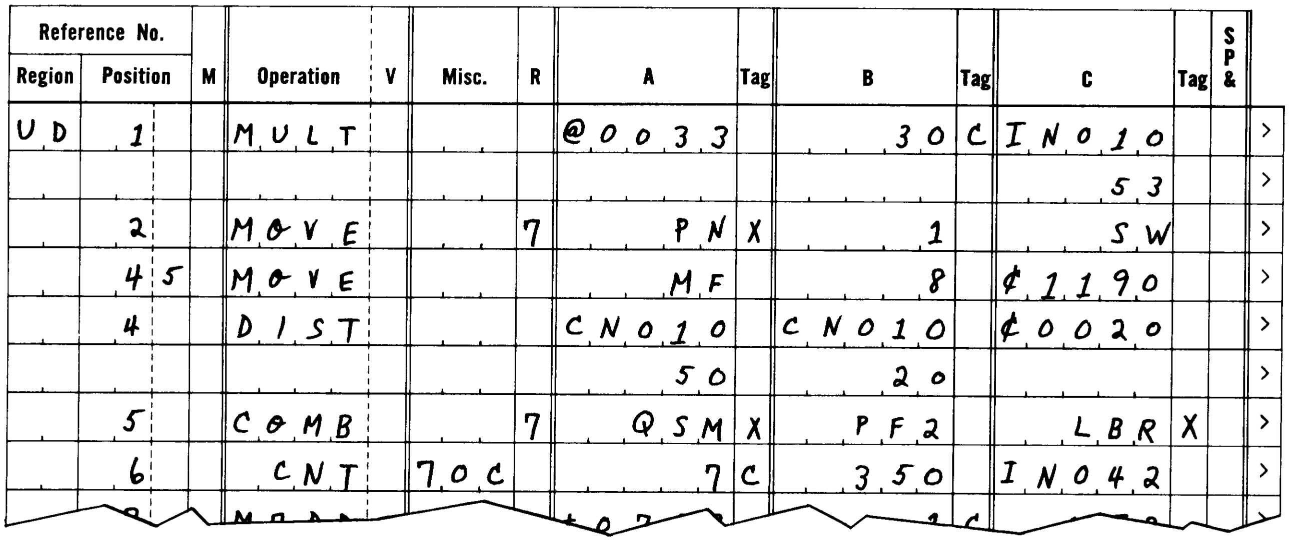
SEQUENCE OF REGIONS WITHIN A PROGRAM:

The Neat **ASSEMBLY** normally assembles the Regions in the same sequence as they are presented to it. The sequence of Regions may, however, be unconditionally determined by the use of the Assembly Instruction NEXT (see Appendix, page C-17).

SEQUENCE OF INSTRUCTIONS—CORRECTIONS TO A PROGRAM:

The Neat **ASSEMBLY** will arrange the Instructions within a Region into the numerical sequence of their Position-numbers, regardless of their sequence on the Coding Sheet. Whenever it finds an Instruction whose Reference Number does not belong in the Region where it appears, it places that Instruction into the proper Region. Whenever two or more Instructions have identical Reference Numbers, the Neat **ASSEMBLY** retains the *latest* one, and discards the others. When all the Instructions in a Region have been brought together, they are then sorted into proper numeric sequence.

Suppose that the program for Region UD has been written, as shown below. Note that, after writing the program (but before keypunching) the programmer noticed that Instruction 3 should have followed Instruction 4. In order to make this correction he merely renumbered Instruction 3 as Instruction 4.5 and the Neat **ASSEMBLY** then placed it into the proper position in the sequence of Instructions.

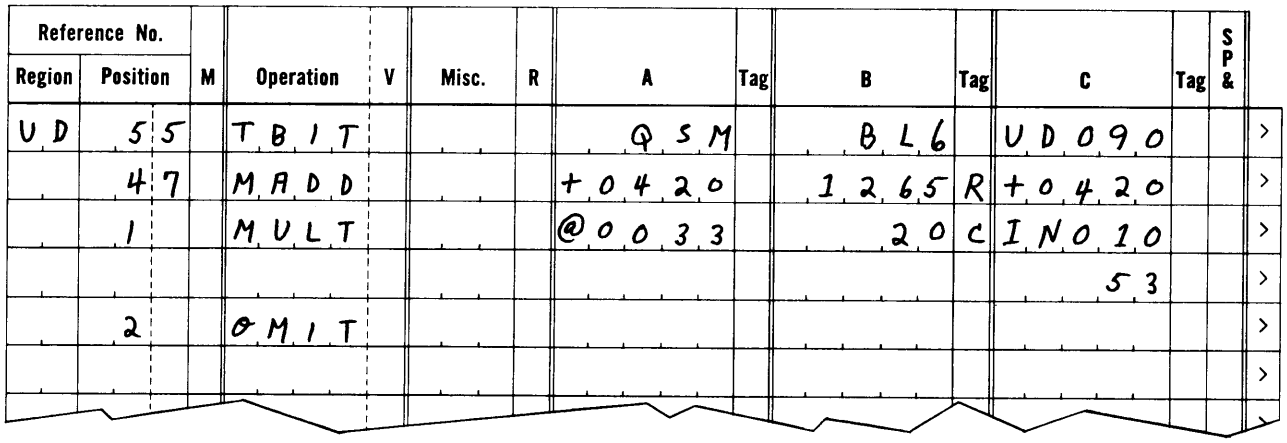


Reference No.														
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&	
UD	1		MULT				@0033		30	C	IN010			>
											53			>
	2		MØVE			7	PN	X	1		SW			>
	4 5		MØVE				MF		8		¢1190			>
	4		DIST				CN010		CN010		¢0020			>
							50		20					>
	5		CØMB			7	QSM	X	PF2		LBR	X		>
	6		CNT		70C		7	C	350		IN042			>

However, after the program had been keypunched, fed to the Neat **ASSEMBLY**, and assembled into a finished program, our programmer found a number of other errors during his code-checking. The corrections which are now needed in Region UD are:

- An additional Instruction is required between 5 and 6.
- An additional Instruction is required immediately following the one which is now numbered 4.5.
- Instruction 1 should call for multiplication by 20 rather than by 30.
- Instruction 2 must be dropped from the program.

The programmer makes up another Coding Sheet, listing the corrections:



Reference No.														
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&	
UD	5 5		TBIT				QSM		BL6		UD090			>
	4 7		MADD				+0420		1265	R	+0420			>
	1		MULT				@0033		20	C	IN010			>
											53			>
	2		ØMIT											>
														>

This sheet is keypunched, and the additional cards placed at the back of the program deck, or the additional paper tape spliced to the end of the previous program tape, as the case may be. Then the entire program is fed to the Neat ***ASSEMBLY*** again, and a new assembled program is automatically prepared. In the new assembly, the sequence of Instructions in Region UD will be:

UD.01.0	MULT	(new version—multiply by 20)
04.0	DIST	
04.5	MOVE	
04.7	MADD	
05.0	COMB	
05.5	TBIT	
06.0	CNT	
07.0	MADD	

—etc.—

DATA DEFINING SHEET:

It has already been mentioned that the programmer makes out a Data Defining Sheet for each kind of Item referred to by the program. Each sheet names an Item by assigning an Item Designator to it, and shows all the information-fields within that Item. Initially, the programmer lists in the Description column all the fields he expects to use, with the additional information provided for his convenience. These Field Descriptions will suffice for the early planning stages of the program, and in fact the list will grow as detailed analysis of the problem indicates the need for additional information within the Item.

As work progresses toward a final program, the specific Item layouts will be drawn, and Field Designators assigned. The Field Designators will be entered on the Data Defining Sheets with their positions (location and field) within their respective Items. The position of a Field within an Item is its "relative address" with respect to the first word of the Item. The sequence in which the Fields are listed need bear no relationship whatever to their positions within the Item, and in fact they will be listed as the need for each of them comes to the programmer's mind. At some convenient time the headings of the sheet will be filled in, and now the sheet is complete, except for Memory Allocation, and Base.

Assigning the Memory Allocation to each Gulp of Items is just a matter of dividing up whatever Memory space is not used by the program. After the complete program has been written, it is easy to make an accurate estimate of the amount of Memory space required for it, since the Neat ***FORMAT*** is one-to-one with Processor Code; then the remaining Memory space is available for data. Assume that the total program for Example 1 required 1052 words; the data area would start in Cell 1053, and 948 words of a 2000-word Main Memory would be available for data.

Referring to the Item layouts of Example 1, which are repeated on page 15, suppose that the programmer chose to place 50 Input Items per 100-word Record; 5 Output Items per 15-word Record; and of course, 1 Master File Item per 18-word Record. He further decided that Input would be handled in Gulps of 3 Records (300 words), Output in Gulps of 13 Records (195 words), and the Master File in Gulps of 25 Records (450 words), using 945 of the 948 words available for data.

Neat DATA DEFINING SHEET National* **NCR 304 DATA PROCESSOR**

FILE NAME________________ FILE DESIGNATOR ☐

ITEM NAME________________ STANDARD ABBREV.________

REMARKS________________ PAGE NO.______FOR THIS I.D.

	DATA	>
ITEM DESIGNATOR		/
BASE		>

ITEM LENGTH______WORDS MEMORY ALLOCATION______WORDS

PREPARED BY________________DATE______

APPROVED BY________________DATE______

FIELD DESIGNATOR		LOCATION	FIELD		DESCRIPTION—COMMENTS	No. of Characters	A or N
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			

X-1742-26 5-1-58 THE NCR Co. *Trade-Mark Reg. U. S. Pat. Off.

These Memory Allocations may now be entered on the Data Defining Sheets, and the Base of each Item assigned. Let us say that the programmer wishes the Output Items to be at the beginning of the data area; then the Input Items; and finally the Master File Items. He will first assign Base zero to Output, meaning that the Output Gulp will start zero words after the Origin of the data area. Then, since the Memory Allocation for Output is 195 words, the Input Items are to start 195 words later in Memory, and they will be assigned Base 195. Next, since the File Items are to start 300 words later than Input, they will be assigned Base 495.

As a precaution against his own arithmetic errors, the programmer will then make out one more Data Defining Sheet, defining Item X, which is to follow the File Items, and which therefore has Base 945. If the programmer *has* made an error, and there is not sufficient Memory space for the 450 words he wishes to allocate to the File Items, the *Neat* ***ASSEMBLY*** will not be able to fit Item X into Memory, and will immediately notify him by means of a printout. This warning will save the programmer from getting into serious trouble during code-checking, as it immediately permits him to reassign Memory Allocations on the basis of accurate information furnished by the *Neat* ***ASSEMBLY***, and then reassemble.

These decisions may be summarized in the following table:

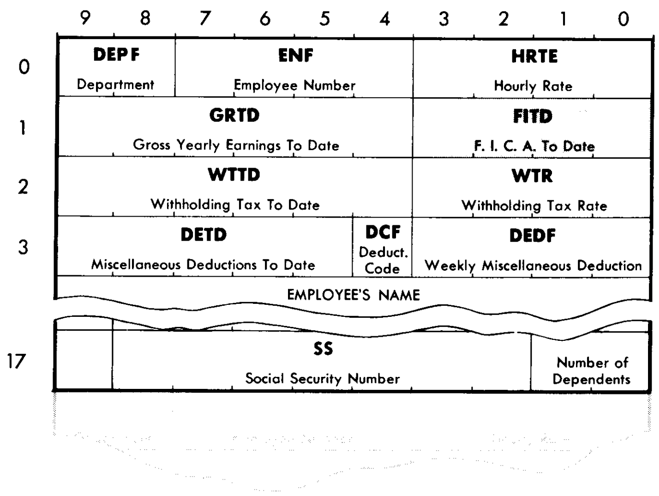
	ITEM DESIGNATOR	ITEM LENGTH	ITEMS PER RECORD	RECORD LENGTH	RECORDS PER GULP	MEMORY ALLOCATION (GULP LENGTH)	BASE	CORRESPONDING FILE DESIGNATOR
Output	**SUM**	3	5	15	13	195	0	**SUM**
Input	**TIME**	2	50	100	3	300	195	**TCS**
File	**EMF**	18	1	18	25	450	495	**EMF**
X	**X**	1	—	—	—	1	945	**—**

While writing the program, the programmer will not be able to complete any of the Magnetic Tape, Paper Tape, Punched Card, or Count Instructions until he has determined how big a Gulp of each kind of Item will be processed. But he usually cannot determine Gulp sizes until the entire program has been written and he knows how much Memory space is available for data. Therefore it is wise to mark all these incomplete Instructions clearly in the margin of the Coding Sheet, and also to keep a list of them. They may then be completed after the Data Defining Sheets have been made up, and if any change in Gulp sizes is later made, these Instructions can be changed to conform.

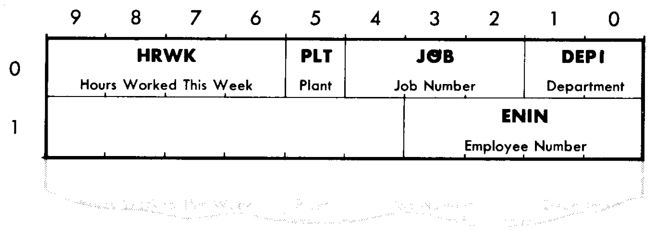
On the other hand, if the programmer has been trained to proper work habits, and proceeds systematically through a sequence of well-documented and increasingly detailed flow charts before reaching for a Coding Sheet, he will be able to estimate the length of his program quite closely, and assign Memory allocations for data, on the basis of his flow charts. In that case he will not have to leave any blanks in his program when he does write it in *Neat* ***FORMAT***. It is also wise, of course, not to use every last Cell of Memory for data, but to leave enough room to permit later changes and corrections in the program without having to reassign Gulp lengths.

The Item layouts and Data Defining Sheets for Example 1 are shown below, and on the following two pages.

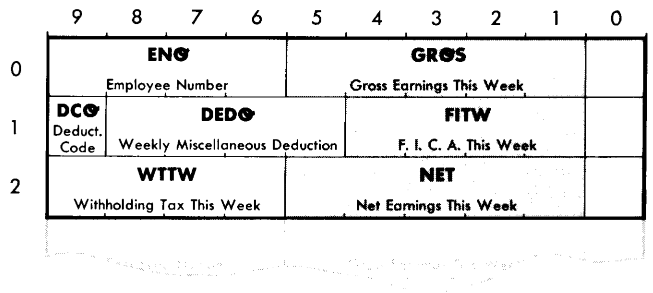
Employee Master File Items

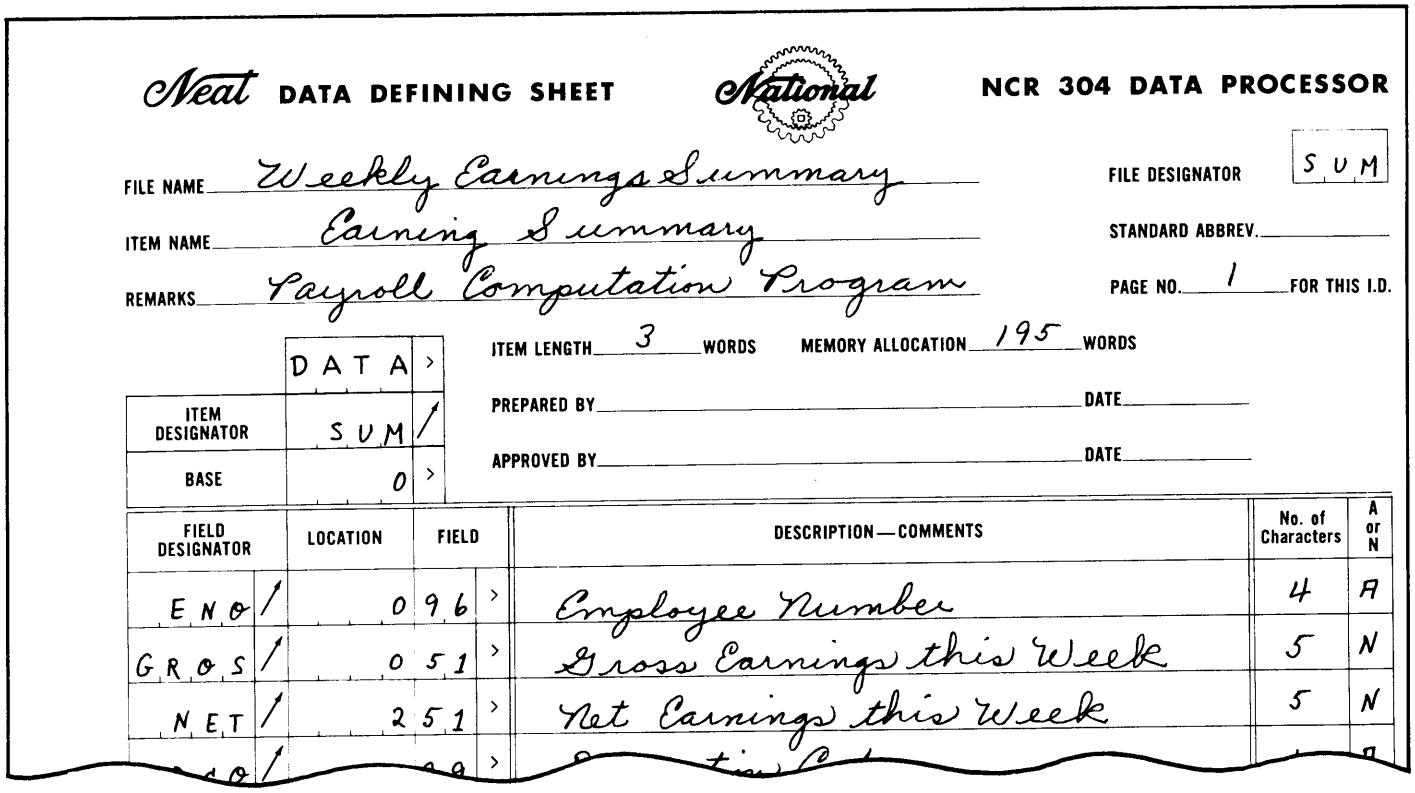


Input Items



Output Items





Neat DATA DEFINING SHEET — National — NCR 304 DATA PROCESSOR

FILE NAME Weekly Earnings Summary — FILE DESIGNATOR SUM

ITEM NAME Earning Summary — STANDARD ABBREV.

REMARKS Payroll Computation Program — PAGE NO. 1 FOR THIS I.D.

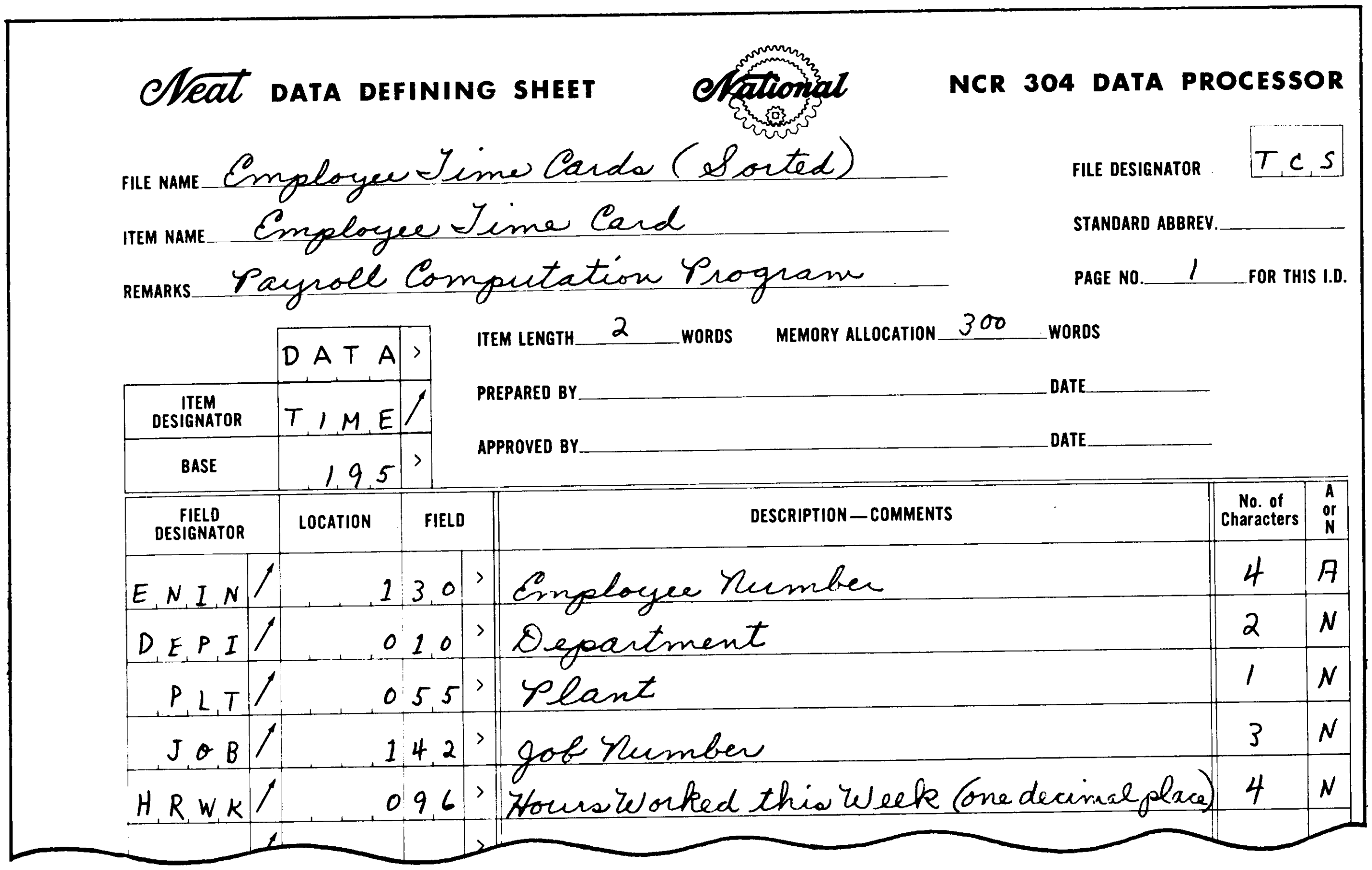
ITEM LENGTH 3 WORDS — MEMORY ALLOCATION 195 WORDS

PREPARED BY ____ DATE ____

APPROVED BY ____ DATE ____

	DATA	>
ITEM DESIGNATOR	SUM	/
BASE	0	>

FIELD DESIGNATOR		LOCATION	FIELD		DESCRIPTION—COMMENTS	No. of Characters	A or N
ENO	/		096	>	Employee Number	4	A
GROS	/		051	>	Gross Earnings this Week	5	N
NET	/		251	>	Net Earnings this Week	5	N



Neat DATA DEFINING SHEET — National — NCR 304 DATA PROCESSOR

FILE NAME Employee Time Cards (Sorted) — FILE DESIGNATOR TCS

ITEM NAME Employee Time Card — STANDARD ABBREV.

REMARKS Payroll Computation Program — PAGE NO. 1 FOR THIS I.D.

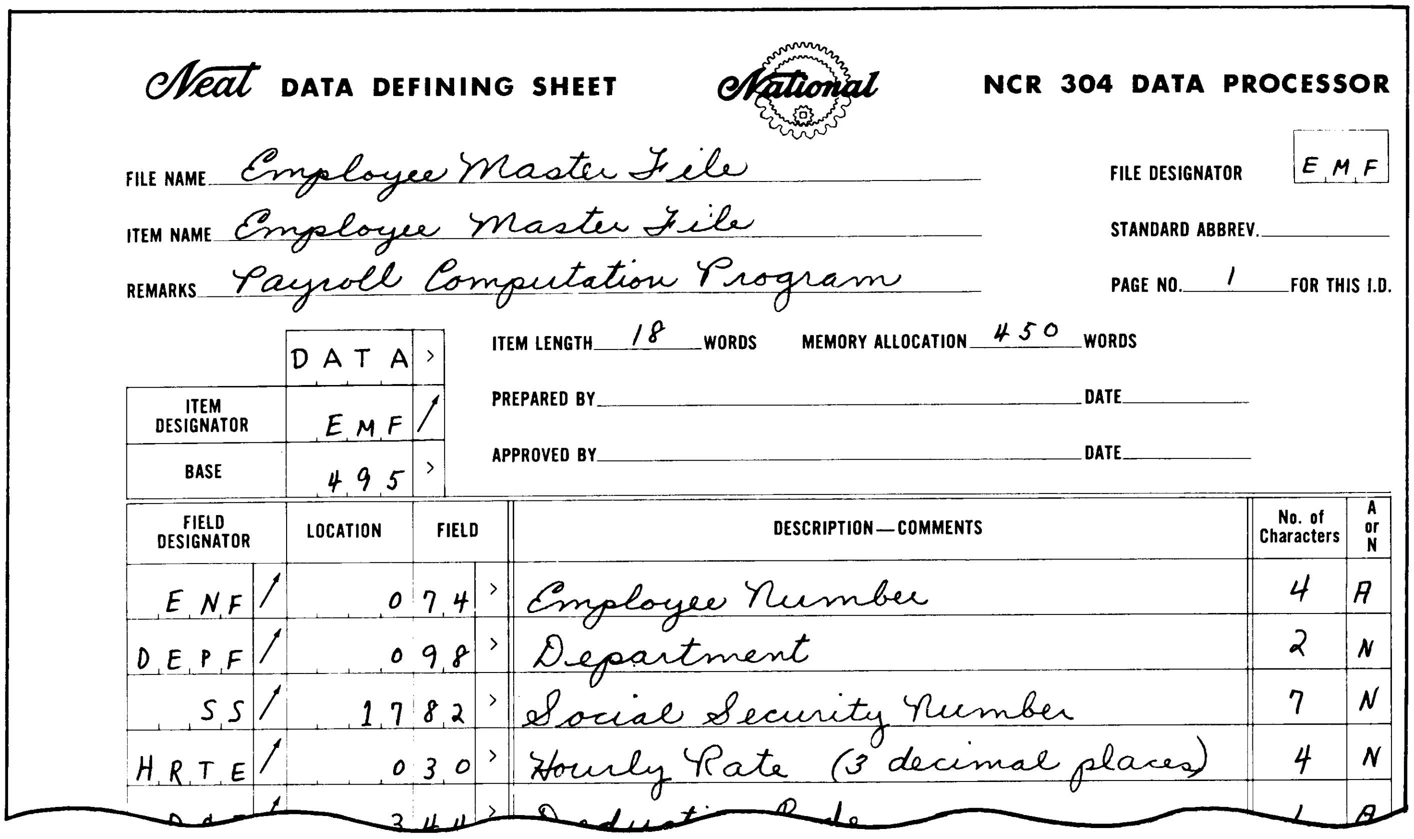
ITEM LENGTH 2 WORDS — MEMORY ALLOCATION 300 WORDS

PREPARED BY ____ DATE ____

APPROVED BY ____ DATE ____

	DATA	>
ITEM DESIGNATOR	TIME	/
BASE	195	>

FIELD DESIGNATOR		LOCATION	FIELD		DESCRIPTION—COMMENTS	No. of Characters	A or N
ENIN	/		130	>	Employee Number	4	A
DEPI	/		010	>	Department	2	N
PLT	/		055	>	Plant	1	N
JOB	/		142	>	Job Number	3	N
HRWK	/		096	>	Hours Worked this Week (one decimal place)	4	N

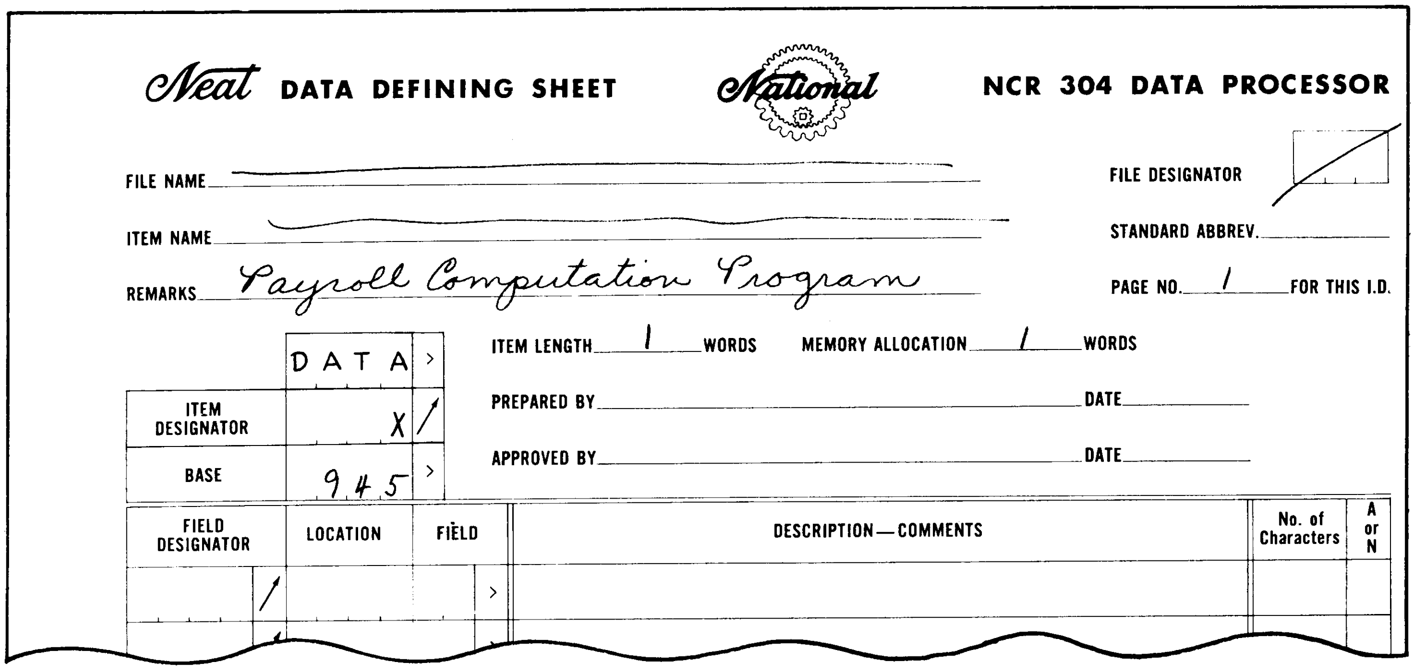


Neat DATA DEFINING SHEET National **NCR 304 DATA PROCESSOR**

FILE NAME	Employee Master File
FILE DESIGNATOR	E M F
ITEM NAME	Employee Master File
STANDARD ABBREV.	
REMARKS	Payroll Computation Program
PAGE NO.	1 FOR THIS I.D.
ITEM LENGTH	18 WORDS
MEMORY ALLOCATION	450 WORDS
PREPARED BY	
DATE	
APPROVED BY	
DATE	

	DATA >
ITEM DESIGNATOR	E M F /
BASE	4 9 5 >

FIELD DESIGNATOR	LOCATION	FIELD	DESCRIPTION—COMMENTS	No. of Characters	A or N
E N F /	0 7 4	>	Employee Number	4	A
D E P F /	0 9 8	>	Department	2	N
S S /	1 7 8 2	>	Social Security Number	7	N
H R T E /	0 3 0	>	Hourly Rate (3 decimal places)	4	N
[illegible] /	3 4 4	>	[illegible]	1	A



Neat DATA DEFINING SHEET National **NCR 304 DATA PROCESSOR**

FILE NAME	
FILE DESIGNATOR	
ITEM NAME	
STANDARD ABBREV.	
REMARKS	Payroll Computation Program
PAGE NO.	1 FOR THIS I.D.
ITEM LENGTH	1 WORDS
MEMORY ALLOCATION	1 WORDS
PREPARED BY	
DATE	
APPROVED BY	
DATE	

	DATA >
ITEM DESIGNATOR	X /
BASE	9 4 5 >

FIELD DESIGNATOR	LOCATION	FIELD	DESCRIPTION—COMMENTS	No. of Characters	A or N
/		>			

ASSEMBLY OF DATA REFERENCES:

After the *Neat* **ASSEMBLY** has put the Instructions of the program into proper sequence, and assigned a precise Memory Address to each of them, it is ready to cross-index the data references. In the previous example, we assumed that the program required 1052 words of Memory, and therefore the data area began at Cell 1053.

Item Designator **SUM** is specified as having Base zero, relative to the Origin of data area, and therefore the first word of the first **SUM** Item will be in Cell 1053. The *Neat* **ASSEMBLY** then calculates the Memory address of each Field Designator in the Item by adding:

> Origin of the data area,
> plus Base of the Item, relative to the Origin,
> plus Location of the Field, within the Item.

Thus Field Designator **ENØ** corresponds to Cell 1053, and of course the 96-field of that Cell. **GRØS** corresponds to Cell 1053:51, **NET** to 1055:51, etc.

When all the Fields in the first Item Designator have been assigned Memory addresses, the *Neat* **ASSEMBLY** proceeds to the next Item Designator which, in this example, is **TIME** with Base 195. Therefore the first word of the first Item in the Gulp of Employee Time Cards will be in Cell 1248, and within that Item, Field Designator **ENIN** corresponds to Cell 1249 (1053 + 195 + 1 = 1249), the 30-field. And so on, until every Field Designator in every Item has been assigned a Memory Address. The Item Designators, too, are treated as though they were actually Field Designators, each referring to the 90-field of the first word of the Item, and they are added to the list.

All the Item Designators and Field Designators are then sorted into a single alphanumeric sequence to form a table. The *Neat* **ASSEMBLY** looks up every program data reference in this table to find the actual Memory address, and plants the address into the Instruction referring to that data.

Note that an Item Designator and its Field Designators are defined only with respect to the first Item in the Gulp. This corresponds to the fact that in the final program, all data references are coded with respect to this first Item, and references to succeeding Items in the Gulp are made by means of Index Register modification of the Instructions.

FILE SPECIFICATION SHEET:

The programmer fills out a File Specification Sheet for each File used in the program. For this purpose, any body of information recorded on Magnetic Tape is regarded as a "File". The Files used in a Payroll Computation Program (such as is simplified in Example 1) might be:

> Sorted Time Cards (source tape only);
> Employee Master File (both source and destination tape);
> Earnings Summaries (destination tape only);
> Exceptions and Statistics (destination tape only).

The first information to be entered on the File Specification Sheet consists of the File-Table Number, the File Designator, and the Index Register which is to control the Handler Assignments for that File. Each File used in the program is represented by a File-Table which the *Neat* **ASSEMBLY** sets up as part of the assembled program. Each File-Table contains the information given on the File Specification Sheet for that File, such as the Controller and Handler assignments for the File, and the alternation of Handlers for successive reels of the

Neat FILE SPECIFICATION SHEET — National* — NCR 304 DATA PROCESSOR

FILE NAME ______________________ FILE DESIGNATOR [ ]

PROGRAM ______________________ PROGRAM DESIGNATOR [ ]

PREPARED BY ______________ DATE ______

CHECKED BY ______________ DATE ______

F I L E >

[ ] /	File-Table Number. (1—9)
[ ] /	File Designator. (*At least one character must be a letter of the alphabet*)
[ ] /	**INDEX REGISTER** assigned to this File. (1—9) (*If none, enter* N)
Co Ph Ah [ ] /	**SOURCE TAPES** for this File (*Enter Controller, Primary Handler, Alternate Handler. If no Source Tapes, enter* N)
Co Ph Ah [ ] /	**DESTINATION TAPES** for this File (*Enter Controller, Primary Handler, Alternate Handler. If no destination Tapes, enter* N)
[ ] /	Is automatic **SETUP** desired on this file? (*Enter* Y *for Yes,* N *for No*)
[ ] /	Length of the longest Record in this File.
[ ] /	Branch Address for **BUSY** Exit. (*If no branch desired, enter* STICK)
[ ] /	Branch Address for **NON-EQUAL STOP** Exit. (*If this Exit not expected, enter* N)
[ ] /	Branch Address for **CRM** Exit. (*If this Exit not expected, enter* N)
[ ] /	Branch Address for Programmer Intervention during **END-OF-TAPE.** (*If no intervention desired, enter* N)
[ ] /	Is **USE LOCKOUT** desired on Rewind? (*Enter* Y *for Yes,* N *for No*)
[ ] /	Shall **RESCUE POINTS** be established on the Destination Tapes of this File? (*Enter* Y *for Yes,* N *for No*)
[ ] /	Relative Position (0—7) of the **UNIQUE WORD** within each Record. (*If no Rescue Points on any file in this Program, enter* N)
[ ] /	Number of periods during which this File is to be protected by expiration-check. (*If File is on M-period, enter* M *and two digits*)
[ ] >	Number of periods prior to this run, when Source-Tape was made. (*If File is on M-period, enter* M *and two digits*)

X-1742-13 6-1-58 The NCR Co. *Trade-Mark Reg. U. S. Pat. Off.

File, all of which are usually controlled by the A-syllable of an Index Register. The File-Table also tallies successive reels of the File, to be sure the reels are processed in proper sequence, and it is consulted by the label-checking functions of STEP. The format of the File-Tables is shown in Appendix B.

The File-Tables are numbered from 1 to 9, depending on the number of Files used in the program, and one File-Table Number is intimately associated, within a given program, with each File and File Designator. In the assembled program, every Magnetic Tape Instruction carries the designation **F** in the 55-field of its second word; this is the File-Table Number corresponding to the particular File which is mounted on the Handler(s) addressed by that Instruction. This is automatically planted in every Magnetic Tape Instruction by the *Neat* ***ASSEMBLY.***

The assignments for the Payroll Computation Program would be:

FILE TABLE No.	FILE DESIGNATOR	FILE NAME	INDEX REG.
1	T C S	Sorted Time Cards	1
2	E M F	Employee Master File	2
3	S U M	Earnings Summary	1
4	E X	Exceptions + Statistics	3
5			
6			

It will be seen that, although each file must have its own File-Table, a file used as source only, may be assigned the same Index Register as some other file used as destination only, provided both files are on the same Controller. Also, only the A-syllable of each Index Register is involved in these assignments.

The programmer next assigns Controller, Primary Handler and Alternate Handler for the Source Tapes and Destination Tapes of the file. If any file consists of only one reel of tape, and therefore no alternation is required, the same Handler-number will be entered as both Primary and Alternate.

The next question on the File Specification Sheet deals with SET FILE, which is one of the first steps any program must perform. This involves:

- Preset the appropriate Index Register for the Primary Source Handler and/or Primary Destination Handler.
- Label-check the first reel of Source Tape to be sure it is:
  The correct file;
  The first reel of that file;
  The correct day's recording of that file;
- If this is a file on which Rescue Points have been established, Index Forward over the Memory Dump which follows the Label-Record.
- Label-check the first reel of Destination Tape to be sure it is obsolete, and may safely be written on.
- Record a new label on the Destination Tape to identify the information which is about to be recorded on it.

These operations are performed as part of the Executive functions of STEP, and are normally performed automatically on each file at the beginning of each day's run, before STEP permits the program itself to begin operating. However the programmer may, instead, choose the moment at which these functions are to be performed on any file by answering "No" to the question "Is automatic SETUP desired on this file" and by placing the Pseudo-Instructions SET:S and/or SET:D into his program at the desired place (see Appendix, page C-16).

The remaining questions on the File Specification Sheet are discussed in Appendix A.

FILES, FILE DESIGNATORS, and ITEM DESIGNATORS:

The distinctions among a File, the Items in the File, and the Item Designators associated with those Items, are important and must be clearly understood.

A File, naturally, consists of Items, each of which has its specific format, as displayed in the Item layouts, and the information on a Data Defining Sheet will initially be drawn up as part of planning a File. Once an Item is in Memory, however, it loses all connection with the File, and the Item Designator is just a reference to a particular Memory location. Similarly, the File Designator is, as far as the program is concerned, nothing but a convenient way of expressing the Controller and Handler to be addressed by some Magnetic Tape Instruction. The programmer, of course, always thinks in terms of the File which is mounted on a particular Handler, and he thinks of the Items in it as belonging to that File. Yet in the actual mechanics of coding, when operating on Items which have been brought into Memory, or which are being built up in Memory, he must remember that Item Designators are *merely* Memory addresses, and File Designators are *merely* Controller-Handler designations.

Therefore, if an Item is moved from one Memory area to another, a new Item Designator (with its complement of Field Designators) must be defined for the new area. Of course, only those Fields which are referred to in each area need receive Field Designators within that corresponding Item Designator, and in many cases no Field Designators at all will be defined for one of the areas.

Further, some Items will exist without reference to any File at all, as when the programmer sets up an area of Memory as Working Storage. He assigns an Item Designator to this area, and sets up as many Field Designators as may be convenient.

The connection between an Item and a File is established by the Magnetic Tape Instructions, as when reading Input in the preceding example:

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
Region	Position												
			READ		3		TCS		TIME		300		

This Instruction says READ 3 Records
—from the Handler corresponding to File Designator **TCS**
—into the Memory Cell corresponding to Item Designator **TIME**
—filling not over 300 words of Memory.

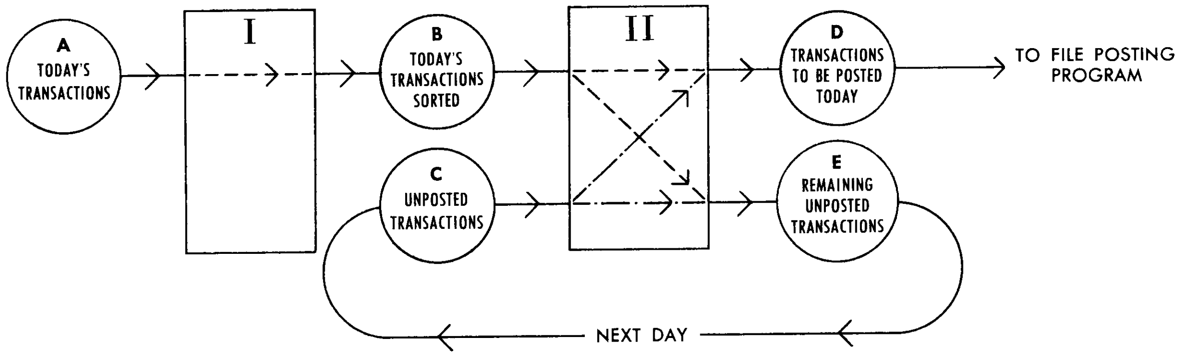
It will often be convenient to use the same code for a File Designator and for a corresponding Item Designator, *provided* the programmer never forgets that he is thereby using the same name for two different things. This has been done for the Output in the preceding example, and the Magnetic Tape Instruction for writing Output would be:

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP &
			WT	F	13		SUM		SUM		15		

This Instruction says WRITE (Fixed-Length) 13 Records
—on the Handler corresponding to File Designator **SUM** (see File Specification Sheet)
—from the Memory Cell corresponding to Item Designator **SUM** (See Data Defining Sheet)
—each Record to be 15 words long.

Thus the Item Designators lead a "double life". Internally they are Memory addresses; while externally they have, in the programmer's mind, an association with their corresponding Files.

Suppose that the first two programs of a daily Processor run are:


A
TODAY'S TRANSACTIONS
I
B
TODAY'S TRANSACTIONS SORTED
C
UNPOSTED TRANSACTIONS
II
D
TRANSACTIONS TO BE POSTED TODAY
E
REMAINING UNPOSTED TRANSACTIONS
TO FILE POSTING PROGRAM
NEXT DAY

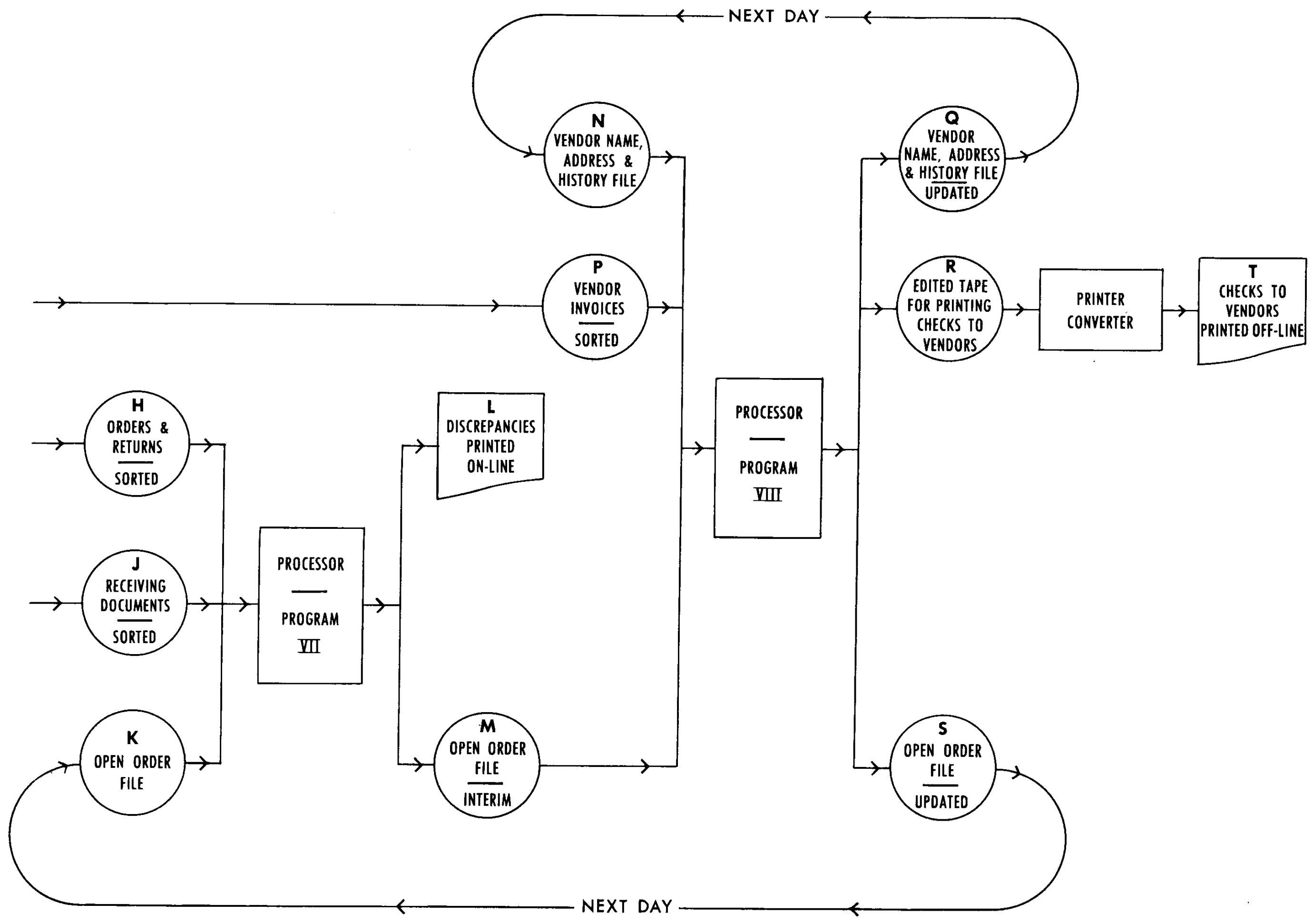
Program II points up the necessity of thinking of an Item Designator within a program *solely* as a Memory Address. From the external point of view, all the Items involved in this program are "transactions", and they are all the same kind of thing, as they are shuffled around among the four tapes. But within the program, four Memory areas (and therefore, four Item Designators) will be set up for the transactions:

Read-in area from tape B,
Read-in area from tape C,
Write-out area for tape D,
Write-out area for tape E.

As a reflection of the "double life" which Items lead, provision is made on the Data Defining Sheet to assign an informal standard abbreviation for the Item, if desired. When mentioning an Item in its "external" capacity, it will often be convenient to refer to it by this standard abbreviation. In this example, there will be occasion to speak in this informal manner of the flow of "transactions" throughout the entire Processor daily run, and TRANS would be an appropriate standard abbreviation for this purpose.

The File Designators lead a similar "double life". Internally for the purpose of the *Neat* ***ASSEMBLY***, a File Designator is a Controller-Handler designation; but externally, it serves as a File Title within the label-record which is recorded on every reel of Magnetic Tape, and which STEP uses to guarantee that the correct tapes have been mounted on the Handlers for each program.

To clarify the function of the File Designator in its "external" capacity of File Title, the following illustration shows the last two Processor programs in an Accounts Payable operation. The letter-designations of each tape merely permit identification within this illustration, and must not be confused with the File Designators which the programmer will assign.



NEXT DAY
N VENDOR NAME, ADDRESS & HISTORY FILE
Q VENDOR NAME, ADDRESS & HISTORY FILE UPDATED
P VENDOR INVOICES SORTED
R EDITED TAPE FOR PRINTING CHECKS TO VENDORS
PRINTER CONVERTER
T CHECKS TO VENDORS PRINTED OFF-LINE
H ORDERS & RETURNS SORTED
L DISCREPANCIES PRINTED ON-LINE
PROCESSOR PROGRAM VIII
J RECEIVING DOCUMENTS SORTED
PROCESSOR PROGRAM VII
K OPEN ORDER FILE
M OPEN ORDER FILE INTERIM
S OPEN ORDER FILE UPDATED
NEXT DAY

Tapes H, J, P and R are all separate Files, and each must have a unique File Title. But note that tape Q becomes tape N the following day, and must have the same File Title as tape N; tapes N and Q are, therefore, defined as the source and destination tapes of a single file. They may be distinguished from each other by the fact that their labels show them to have been recorded on different days.

Similarly, tape S becomes tape K the following day; it must have the same File Title, and must be considered the same File, as tape K. Tape M, however, *is not* the same File as K and S. Tape K is the Open Order File updated for both merchandise and payments as of yesterday; tape S is the same File updated for both merchandise and payments as of today. But tape M is updated for merchandise as of *today*, and for payments as of *yesterday*; it can never serve as tape K, and in fact it is essential to label it in such a way that it is impossible, by accident, to use it as tape K. This is achieved by considering it as a different File, with a different File Title.

Once this *principle* has been understood, however, it must be recognized that this approach may involve us in a contradiction, or may at least cause some inconvenience. It will clearly be more convenient, when writing program VII, to consider Tapes K and M as the same File with the same File Designator; and when writing program VIII, to consider Tapes M and S as the same File with the same File Designator. Furthermore, if COPY is used in either program, the two tapes involved in the COPY *must* have the same File Designator.

This difficulty is resolved by pointing out that the File Designator will *usually* serve as the File Title for both source and destination tapes, but there are occasions (like this one) when it will not so serve. In this example, the programmer will use the same *File Designator* for Tapes K, M and S in programs VII and VIII, but will use some other 3-character code as the *File Title* for Tape M (destination tape) in program VII, and for Tape M (source tape) in program VIII.

Since the File Designators will usually serve as File Titles, the *Neat* **ASSEMBLY** stores each File Designator in the appropriate File-Table as the File Title of the source tape, and as the File Title of the destination tape. In a case such as this one, when certain tapes require File Titles which are different from their File Designators used during the Assembly, the programmer may change the File Titles after the program has been assembled, before recording it in the Program-Library.

This subject is discussed at greater length in Appendix B, in connection with the label-checking functions of STEP.

APPENDIX A

Neat SHEETS

APPENDIX A

PROGRAM HEADER SHEET

Neat PROGRAM HEADER SHEET

National*

NCR 304 DATA PROCESSOR

PROGRAM__________

PROGRAM DESIGNATOR

REMARKS__________

PREPARED BY__________ DATE______

CHECKED BY__________ DATE______

FILES USED IN THIS PROGRAM

FILE TABLE No.	FILE DESIGNATOR	FILE NAME	INDEX REG.
1			
2			
3			
4			
5			
6			
7			
8			
9			

H E A D >

Program Designator. *(Name and Identification Number)*

Memory Address for Program Origin. *(If unspecified, enter* NEAT*)*

Memory Address for Data Origin. *(If unspecified, enter* NEAT*)*

OPTION 1 Shall the assembled program be punched? (i.e. on Paper Tape or Cards) *(Enter* Y *for Yes,* N *for No)*

OPTION 2 Shall the assembled program be recorded on Magnetic Tape? *(Enter* Y *for Yes,* N *for No)*

OPTION 3 Will the Printer be on-line or off-line during the assembly? *(Enter* ØN *or* ØFF*)*

OPTION 4 Is this a Main Program or an Overlay? *(Enter* P *or* Ø*)*

If this is a Main Program, what is the START Address? *(If Overlay, enter* Ø*)*

OPTION 5 Is Standard Rescue Point Subroutine desired with this Program? *(Enter* Y *for Yes,* N *for No)*

X-1742-17 6-1-58 The NCR Co.

*Trade-Mark Reg. U. S. Pat. Off.

PROGRAM HEADER SHEET

One Header Sheet will be filled out for each program to be assembled; the information on it will be keypunched into cards or paper tape, and will be input to the *Neat* **ASSEMBLY** along with the information from the File Specification Sheets, Data Defining Sheets and Coding Sheets.

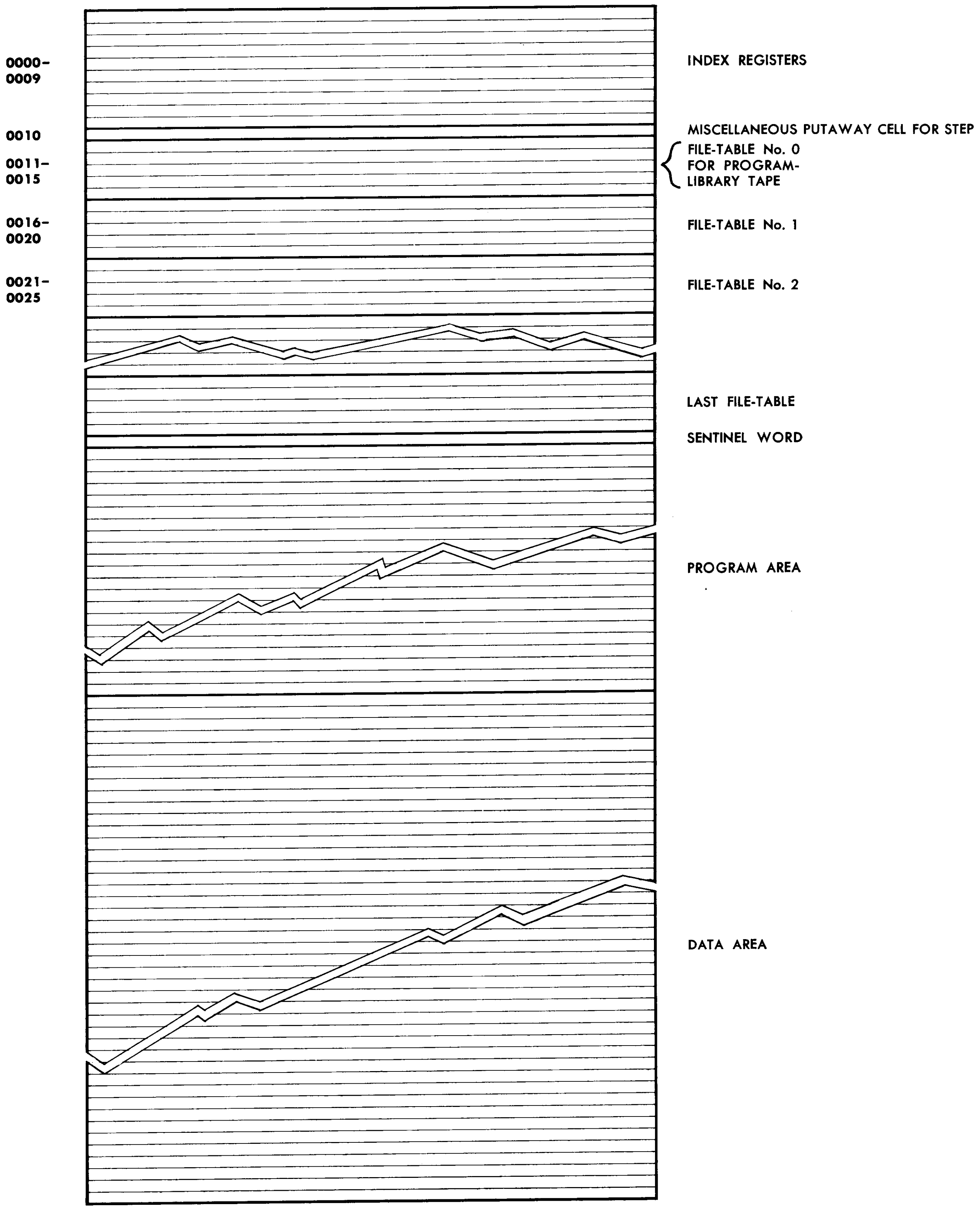
The upper portion of the Program Header Sheet provides for entries to identify the program, and to summarize the assignment of File-Tables and of the Index Registers which control the alternation of Handlers for each File used in the program. The lower portion of the sheet, containing the information to be keypunched, specifies certain options which may be exercised to govern the assembly process.

The operator uses the Console Option Switches, and manual entries through the Console Typewriter, to inform the *Neat* **ASSEMBLY** whether input is from paper tape, punched cards, or magnetic tape; which Handlers are available to the assembly; etc. These details are described in the *Neat* Operating Manual.

In the assembly, Cells 0000 - 0009 are reserved for the Index Registers; Cell 0010 is reserved as a special putaway address used by STEP; Cells 0011 - 0015 are reserved for File-Table No. 0, which governs use of the Program-Library Tape mounted on Controller No. 0, Handler No. 0; and starting in Cell 0016 are as many File-Tables as are specified for the program, occupying 5 words each. Even though some File-Table Numbers may be unused, the *Neat* **ASSEMBLY** will set aside 5 words for each Table, up to and including the highest File-Table Number assigned. Thus, for example, if File-Table No. 4 is assigned to a File in the program, it will always appear in Cells 0031 - 0035, whether or not File-Tables 1, 2 and 3 are being used. After the last File-Table is a sentinel-word.

Normally, the program is stored immediately after the highest-numbered File-Table, and the data area falls immediately after the last word of the program. But these dispositions may be changed by the entries on the Header Sheet.

ASSEMBLY OF A PROGRAM IN MEMORY



PROGRAM DESIGNATOR:

This consists of an 8-character alphanumeric Name, and a 2-digit numeric Number, identifying the program.

The designation may be Program Name, with the numbers identifying successive corrections and revisions of the program, and also different overlays used within the program, for example:

PAY IN	04	Input Regimentation (4th version)
PAYSORT1	01	First Sorting Operation (1st version)
PAY FILE	10	Posting Employee Master File (10th version)
PAY FILE	52	Overlay A (2nd version)
PAY FILE	67	Overlay B (7th version)
PAY FILE	71	Overlay C (1st version)
PAYSORT2	01	Second Sorting Operation (1st version)
....	..	and so on. ....

Another possible scheme would be the assignment of a Job Name, and Program Number within the job, with successive versions of any program keeping the same Designator, for example:

PAYROLL	10	Input Regimentation
PAYROLL	20	First Sorting Operation
PAYROLL	30	Posting Employee Master File
PAYROLL	31	Overlay A
PAYROLL	32	Overlay B
PAYROLL	33	Overlay C
PAYROLL	40	Second Sorting Operation
....	..	and so on. ....

Since the assignment of Program Designator is completely arbitrary, these two schemes are merely illustrative of the available choices, and any scheme may be adopted which suits the convenience of a particular installation.

MEMORY ADDRESS FOR PROGRAM ORIGIN:

Instead of permitting the *Neat* **ASSEMBLY** to place the first word of the program immediately after the last File-Table, or immediately after the Standard Rescue Point Subroutine if it is used (Option 5), the programmer may specify the precise Memory address for the Origin of the program. This facility is particularly useful when assembling an overlay which is to be used with a main program. Overlays are discussed on page A-8.

If a Memory address is specified for program origin, it will, of course, be expressed as a 4-digit number, since *there are no Address-Type Numbers* in *Neat* **FORMAT.**

MEMORY ADDRESS FOR DATA ORIGIN:

In similar fashion, the programmer may choose to specify (as a 4-digit number) the precise Memory address which shall correspond to Base zero of the data area.

SHALL THE ASSEMBLED PROGRAM BE PUNCHED?

If the answer is "Yes" the Neat **ASSEMBLY** will punch the final program into paper tape on-line if the Neat **FORMAT** was input from paper tape; or it will record the program on magnetic tape for off-line card punching if the input was from cards.

SHALL THE ASSEMBLED PROGRAM BE RECORDED ON MAGNETIC TAPE?

If "Yes" the Neat **ASSEMBLY** will comply. This will not be the Program-Library Tape, however; recording that tape is performed only by the Librarian Program, described in the Neat Operating Manual.

WILL THE PRINTER BE ON-LINE OR OFF-LINE?

When the Neat **ASSEMBLY** is finished, it prints all pertinent information about the program it has just assembled. The major part of this printout consists of lists of all Item Designators and Field Designators, with the actual Memory addresses assigned to them, and then, in parallel columns:

> Each Instruction in the original Neat **FORMAT**, with its Reference Number;
>
> The Processor Instruction into which it has been translated, with its actual Memory address;
>
> Any comments pointing out possible inconsistencies or errors in writing the program.

This printout may be obtained on-line during the assembly, or recorded on magnetic tape for off-line printing later.

IS THIS A MAIN PROGRAM OR AN OVERLAY?

The assembly of overlays is discussed on page A-8.

WHAT IS THE START ADDRESS?

If this is a main program, this address (which is not necessarily the Program Origin) will be stored in the Program Label which is recorded with this program on the Library Tape. When this program is to be run, the executive portion of STEP will bring it into Memory from the Library, and then automatically jump to this address to begin executing the program.

If this is an overlay program, the main program must contain an Instruction to jump to this overlay, after using the executive portion of STEP to bring it into Memory from the Library. (See page A-8).

IS STANDARD RESCUE POINT SUBROUTINE DESIRED?

If, on *any* File Specification Sheet for this program, the RESCUE POINT question is answered "Yes", then this question on the Header Sheet must also be answered "Yes". If no Rescue Points are desired within this program, answer this question "No". Rescue points are discussed on page A-15.

If this question is answered "Yes", the Neat **ASSEMBLY** places the Standard Rescue Point Routine immediately after the File-Tables. The program being assembled then has its origin immediately after this Subroutine, or at the actual Memory address specified, according to the entry on the Program Header Sheet.

CHANGES TO THE HEADER SHEET:

Once the program has been keypunched into paper tape or cards, the programmer may wish to change some of the entries on the Header Sheet. He makes up a new Header Sheet, has it keypunched, and places the additional cards at the front of the program deck, or splices the additional paper tape to the beginning of the previous program tape, as the case may be. The *Neat* ***ASSEMBLY*** will retain only the entries from the first Header-sheet for any program.

OVERLAYS

An overlay is a section of coding which is not part of the main program, but which is brought into Memory whenever it is needed, usually to provide for some exceptional circumstance in the data. When brought into Memory, it is laid over some other section of coding which is no longer needed. Sometimes it will overlie part of the main program; more commonly, the programmer will choose to set aside a portion of Memory, within the main program, just for overlays, using the Pseudo-Instruction SAVE (see page C-10) to reserve the necessary number of Memory Cells.

In writing an overlay program, the programmer uses the same Item Designators and Field Designators as in the main program; if the overlay performs any Magnetic Tape operations, he also uses the same File Designators. (However, if the overlay uses a File to which the main program does not refer, that File Specification Sheet must be included with the main program in order to set up a File-Table). In naming the Regions for the overlay program, he takes care not to duplicate any Region used by the main program.

The main program is assembled first. The printout furnished by the *Neat* ***ASSEMBLY*** will indicate the actual Memory addresses of the first word in the overlay Region, and of the first word in the data area. Then the programmer fills out a Program Header Sheet for the overlay program, entering these two Memory addresses as the Program Origin and the Data Origin, respectively. The input to the *Neat* ***ASSEMBLY*** for the overlay program consists of:

> The overlay's Program Header Sheet;
>
> The same Data Defining Sheets as used for the main program;
>
> If the overlay program performs any Magnetic Tape operation, the same File Specification Sheets as used for the main program;
>
> The Coding Sheets for the overlay program.

The procedure just described takes care of all data references and file references within the overlay program, but the programmer will have taken several additional steps to provide for jumps back and forth between the overlay and the main program.

JUMPS TO THE OVERLAY PROGRAM:

1) Suppose an overlay program has a single entry-point, which is its first Instruction. The main program will contain a jump to the Reference Number which is designated *within the main program* as the beginning of the overlay area. The actual Memory address corresponding to this Reference Number is the Program Origin for the overlay, and when the overlay program has been brought into Memory its first Instruction will occupy that address. The main program, therefore, will have jumped to the overlay's entry-point.

2) If, when the overlay program has been written, its entry-point is not the first Instruction, the programmer need only add one Instruction at the beginning of the overlay program— a GOTO (see page C-9) the overlay's entry-point.

3) But suppose some overlay has several alternative entry-points. Then the programmer begins the overlay program with a Jump Table, containing a separate GOTO for each entry-point. In terms of the Reference Numbers within the main program, the future location of each of these GOTO Instructions is known, and the main program may jump to them as required.

To illustrate this last situation, consider an overlay program which has 5 entry-points. Its first Region will be:

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
Region	Position												
RJ	1		GOTO								R1010		
	2		GOTO								R1072		
	3		GOTO								R9020		
	4		GOTO								S2160		
	5		GOTO								S4010		
	6		NEXT		R1								

See page C-17 for NEXT

Excerpts from the main program are:

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
Region	Position												
OL	1		SAVE		146		ZERO						

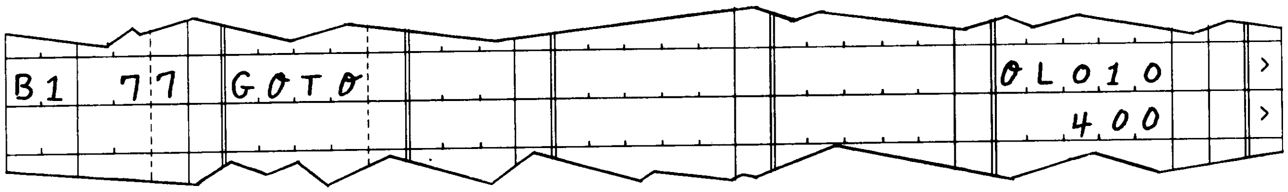
See page C-10 for SAVE.
This reserves 146 Cells for the overlay area.

A2	12		GOTO								OL010		

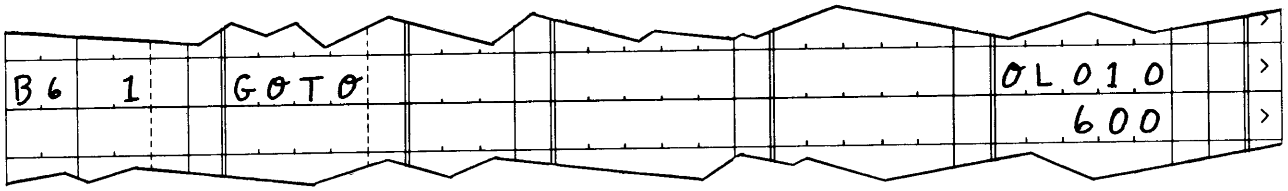
Jumps to RJ.01 which is GOTO R1.01

A2	24		GOTO								OL010		
											200		

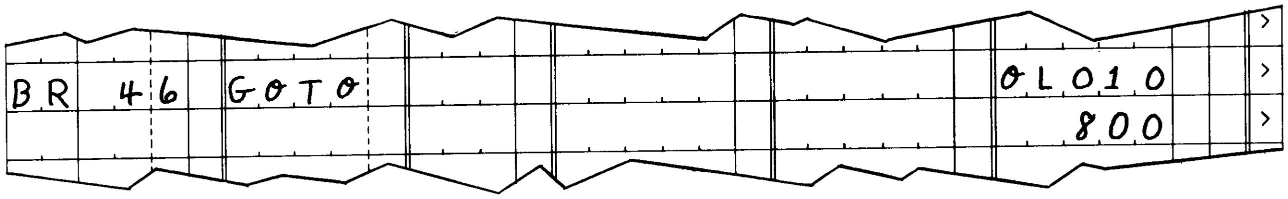
Jumps to RJ.02 which is GOTO R1.07.2



Jumps to RJ.03 which is GOTO R9.02



Jumps to RJ.04 which is GOTO S2.16



Jumps to RJ.05 which is GOTO S4.01

JUMPS BACK TO THE MAIN PROGRAM:

Usually the main program, when jumping to the overlay program, uses minus-V, or some other linking procedure, and the return from the overlay is perfectly straightforward.

If not, then after the main program has been assembled, the actual Memory addresses of the re-entry points are determined from the *Neat* ***ASSEMBLY'S*** printout, and these are entered, as 4-digit Memory addresses, at the proper points in the overlay program.

FILE SPECIFICATION SHEET

Neat **FILE SPECIFICATION SHEET** *National* * **NCR 304 DATA PROCESSOR**

FILE NAME ______ FILE DESIGNATOR

PROGRAM ______ PROGRAM DESIGNATOR

PREPARED BY ______ DATE ______

CHECKED BY ______ DATE ______

F I L E >

⁄	File-Table Number. (1—9)
⁄	File Designator. *(At least one character must be a letter of the alphabet)*
⁄	**INDEX REGISTER** assigned to this File. (1—9) *(If none, enter* **N***)*
Co Ph Ah ⁄	**SOURCE TAPES** for this File *(Enter Controller, Primary Handler, Alternate Handler. If no Source Tapes, enter* **N***)*
Co Ph Ah ⁄	**DESTINATION TAPES** for this File *(Enter Controller, Primary Handler, Alternate Handler. If no destination Tapes, enter* **N***)*
⁄	Is automatic **SETUP** desired on this file? *(Enter* **Y** *for Yes,* **N** *for No)*
⁄	Length of the longest Record in this File.
⁄	Branch Address for **BUSY** Exit. *(If no branch desired, enter* **STICK***)*
⁄	Branch Address for **NON-EQUAL STOP** Exit. *(If this Exit not expected, enter* **N***)*
⁄	Branch Address for **CRM** Exit. *(If this Exit not expected, enter* **N***)*
⁄	Branch Address for Programmer Intervention during **END-OF-TAPE.** *(If no intervention desired, enter* **N***)*
⁄	Is **USE LOCKOUT** desired on Rewind? *(Enter* **Y** *for Yes,* **N** *for No)*
⁄	Shall **RESCUE POINTS** be established on the Destination Tapes of <u>this</u> File? *(Enter* **Y** *for Yes,* **N** *for No)*
⁄	Relative Position (0—7) of the **UNIQUE WORD** within each Record. *(If no Rescue Points on any file in this Program, enter* **N***)*
⁄	Number of periods during which this File is to be protected by expiration-check. *(If File is on M-period, enter* **M** *and two digits)*
>	Number of periods prior to this run, when Source-Tape was made. *(If File is on M-period, enter* **M** *and two digits)*

X-1742-13 6-1-58 The NCR Co. *Trade-Mark Reg. U. S. Pat. Off.

FILE SPECIFICATION SHEET

The heading of the sheet provides reference entries to identify the File and the program. The body of the sheet contains the entries which are to be keypunched and input to the *Neat* ***ASSEMBLY***.

FILE-TABLE NUMBER:

Each File used in a program is assigned a File-Table, which the *Neat* ***ASSEMBLY*** stores in Memory as part of the assembled program, and which contains essentially the information on the File Specification Sheet. The *Neat* ***ASSEMBLY*** automatically plants, in the 55-field of the second word of every Magnetic Tape Instruction, the File-Table Number for the File to which that Instruction refers. In the event of any Tape-Exit condition while the assembled program is operating, STEP picks up the File-Table Number from the Instruction causing the exit, refers to the corresponding File-Table, and conditions itself accordingly.

FILE DESIGNATOR:

A File Designator may be any collection of up to three characters, except that at least one character must be a letter of the alphabet. Whenever the programmer feels that it is convenient to do so, a File Designator may duplicate an Item Designator.

Within any one program, a File Designator is solely an index to Controller, Primary and Alternate Handlers for source and destination tapes. However, the File Designator is also recorded as part of the Label-Record on each reel of magnetic tape, and in that capacity it serves as a File Title which identifies the physical File.

INDEX REGISTER:

Alternation of Handlers for successive reels of source and destination tapes of a File is controlled by the contents of the A-syllable of an Index Register. Index Register #0 is reserved for STEP and for other self-contained subroutines; any of the other Index Registers may be used for controlling the Files. While each File must have its own *File-Table*, one File using only source tapes may be controlled by the same *Index Register* as another File using only destination tapes, provided they are on the same Controller.

If any File is contained on a single reel of tape, and therefore no alternation of Handlers is called for, no Index Register need be specified for that File.

SOURCE TAPES and DESTINATION TAPES:

If source and/or destination tape is contained on a single reel, and therefore no alternation is required, enter the same Handler number for both Primary and Secondary. Then in the event the programmer makes an error, and the File does at some time exceed a single reel, STEP will halt the Processor, after appropriate printout on the Console Typewriter.

Source and destination tapes of any File must be on the same Controller.

IS AUTOMATIC SETUP DESIRED ON THIS FILE?

This significance of this option is discussed on page 20, and on page C-16.

LENGTH OF THE LONGEST RECORD IN THE FILE:

When the assembled program is operating, and an error is detected during a WRITE TAPE operation, the program branches into STEP on the tape-exit condition. STEP then repeats the operation, writing one record at a time, to isolate the record in which the error occured. If that record cannot successfully be written after several tries, STEP assumes that there is a flaw in the tape, and records a special "skip" record, which is as long as the longest record in that file. The writing operation is then resumed, and when it is completed, STEP jumps back to the main program.

This "skip" record will, because of its special characteristics, always be detected whenever that tape is read as a source tape at some later time. This will generate a tape-exit condition, whereupon STEP will identify the "skip" record, ignore it, and allow the main program to proceed.

During recording, STEP tallies the number of "skip" records placed on any reel of tape. If the number exceeds a predetermined figure, STEP prints out that fact on the Console Typewriter, as an indication that it is probably time to discard that tape. The "skip" record is described in detail in Appendix B.

BRANCH ADDRESSES FOR BUSY, NON-EQUAL STOP, CRM:

These are the three branch conditions which the programmer must program for himself; he enters in these spaces the Reference Numbers, or the 4-digit Memory addresses, of his subroutines which handle these conditions. If he does not anticipate the possibility of Non-Equal Stop, or of CRM, he enters "N" for either condition; then if he has made an error, and the condition *does* arise, STEP will halt the Processor, after appropriate printout on the Console Typewriter.

PROGRAMMER INTERVENTION DURING END-OF-TAPE:

The programmer may, if he wishes, intervene with some programming of his own, after detection of end-of-tape on any File, and just before STEP rewinds the tape. The intervention subroutine may perform any work the programmer deems appropriate, except that it must not disturb the contents of Index Register #0, or of Cell ¢00:53, and *it must not perform any Magnetic Tape Operation which can encounter end-of-tape.* The intervention subroutine must end with a Jump to Cell □18, back into STEP for the rewind, alternation of Handlers, and label-check of the next reel.

The programmer may be assured that, even though an error may occur in writing the last Gulp on the reel (and STEP then records these records one at a time, as already described), STEP will still put on that reel all the records which the programmer specified for the Gulp.

Since any intervention will probably be performed by a single subroutine, the following information will be of interest:

- File-Table No. **F** is in Cell ¢25:55.
- If end of source:

Cell ¢00:53	contains "□62".
Cell □29:00	contains "7".
Cell 000:86 } Cell 000:53 }	each contain 5**F**.

- If end of destination:

Cell ¢00:53	contains "□90".
Cell □29:00	contains "6".
Cell 000:86 } Cell 000:53 } Cell 000:20 }	each contain 5**F** + 2.

IS USE LOCKOUT DESIRED ON REWIND?

Every Rewind, of either source or destination tape in this File, will be performed in accordance with the answer to this question.

Remember that this applies only to intermediate Rewinds at the end of successive reels in the File. The final Rewind, at end-of-file, is written by the programmer himself.

RESCUE POINTS:

If desired, STEP will establish a Rescue Point on each new reel of destination tape within this File. In doing so, STEP "marks your place" on every other magnetic tape in the system, and then records the entire contents of the Memory immediately after the label-record on the new destination tape. Then in case of accident the program can always be resumed from the last Rescue Point, without requiring a complete re-run; a Resumption Program is provided for this purpose.

When this destination tape becomes, in its turn, the source tape for some other operation, the end-of-tape subroutine within STEP will recognize the "Memory Dump" immediately after the label on each reel, and will Index the tape over it so that it never enters the Memory during routine operations.

If this question is answered "Yes" for *any* File used in the program, then the Rescue Point question (Option 5) on the Program Header Sheet must also be answered "Yes". The *Neat* ***ASSEMBLY*** will then include the Standard Rescue Point Subroutine as the first Region of the assembled program, immediately after the last File-Table. When the assembled program is operating, and encounters an end-of-destination-tape, STEP will rewind the tape, alternate Handlers, label check the next reel, record a new label on it, and then jump to the Standard Rescue Point Subroutine, which performs a "Memory Dump" on that tape, and then jumps back within STEP for the return link to the operating program.

UNIQUE WORD WITHIN EACH RECORD:

In order that STEP may "mark your place" on each tape in the system when establishing a Rescue Point, every record in every File must contain some word which is unique to that record, within its File. That word will usually be the one containing the Account Number or similar identification.

In establishing a Rescue Point, STEP reads the next record from every tape in the system, and records the contents of its *unique word* in a table which is included in the Memory Dump.

After reading the next record on each tape, that tape is automatically Indexed Backward to its original position, so that routine processing may be resumed.

NUMBER OF PERIODS FOR EXPIRATION-CHECK PROTECTION:

The programmer specifies the number of periods, *including the current one,* during which the label-checking functions of STEP shall not permit any other information to be recorded on any of the destination tapes of this File. When that interval has expired, the tapes are considered obsolete, and may be re-used as destination tapes for some other operation.

Thus if a tape recorded, say, on Monday is to be retained through Wednesday but becomes obsolete by Thursday morning, it must be protected for **3** periods, *including the one during which it was recorded.* If the operation is on a monthly cycle, and the destination tapes are to be held for eleven additional months, they are to be protected for **M12** periods.

NUMBER OF PERIODS PRIOR TO THIS RUN, WHEN SOURCE TAPE WAS MADE:

If the source tape for this program was recorded the previous day, it was recorded **1** period ago. If it is on a weekly cycle, it was recorded **5** periods ago (5-day week), or **6** periods ago (6-day week). If the operation is on an M-cycle, and is being used 3 M-periods after it was recorded, then it was recorded **M03** periods ago.

CHANGES TO THE FILE SPECIFICATION SHEET:

Once the program has been keypunched into paper tape or cards, the programmer may wish to change some of the entries on the File Specification Sheet. He makes up a new Sheet, has it keypunched, and places the additional cards at the back of the program deck, or splices the additional paper tape to the end of the previous program tape, as the case may be. The *Neat* ***ASSEMBLY*** will discard the entries from the first File Specification Sheet for any File-Table, and retain those from the latest one it receives for that File-Table.

DATA DEFINING SHEET

Neat **DATA DEFINING SHEET** *National** **NCR 304 DATA PROCESSOR**

FILE NAME______________________________ FILE DESIGNATOR [ ]

ITEM NAME______________________________ STANDARD ABBREV.__________

REMARKS______________________________ PAGE NO.__________FOR THIS I.D.

	D A T A	>
ITEM DESIGNATOR		/
BASE		>

ITEM LENGTH__________WORDS MEMORY ALLOCATION__________WORDS

PREPARED BY______________________________DATE__________

APPROVED BY______________________________DATE__________

FIELD DESIGNATOR		LOCATION	FIELD		DESCRIPTION—COMMENTS	No. of Characters	A or N
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			
	/			>			

X-1742-26 5-1-58 THE NCR Co. *Trade-Mark Reg. U. S. Pat. Off.

DATA DEFINING SHEET

The heading of the sheet, and the rightmost three columns of the body of the sheet, provide reference entries for the programmer's convenience. Those entries which are keypunched, and input to the Neat ***ASSEMBLY***, are briefly defined below.

ITEM DESIGNATOR:

An Item Designator may be any collection of up to four characters, except that at least one character must be a letter of the alphabet, and no Item Designator may duplicate any Field Designator.

BASE:

The "Relative address" of the first word of the first of these Items in the Gulp, relative to the Data Origin.

FIELD DESIGNATOR:

A Field Designator may be any collection of up to four characters, except that at least one character must be a letter of the alphabet, and no Field Designator may duplicate any Item Designator.

LOCATION:

The "relative address" of the word containing this Field, relative to the first word of the Item.

FIELD:

The partial-word designators for this Field.

DEFINING DATA-FIELDS:

Item Designators and Field Designators are "dictionary entries", from which the Neat ***ASSEMBLY*** obtains the actual Memory addresses for the data-fields named within the program.

Therefore, it is not necessary that the Field Designators cover every field within an Item; any fields which are not referred to within a particular program need not be defined for that program. The programmer may also prefer not to define some of the infrequently-used fields, but to refer to them by their relative positions with respect to some other field which is defined. This was done in Example 2, page 6, and was commented on at that time.

Similarly, the programmer may define *overlapping* Field Designators to whatever extent he finds convenient. In fact, he will often wish to use the same Memory area, at different points in the program, for completely different Items and need merely set up two or more Item Designators, *all with the same Base,* each having its full complement of Field Designators.

CHANGES TO DATA DEFINING SHEETS:

Once the program has been keypunched into paper tape or cards, the programmer may wish to change some of the data definitions. He makes up a new Data Defining Sheet for each Item which is to be changed or added, has it keypunched, and places the additional cards at the back of the program deck, or splices the additional paper tape to the end of the previous program tape, as the case may be.

If a new Data Defining Sheet names an Item Designator which did not previously appear, the *Neat* ***ASSEMBLY*** will add it, with its Field Designators, to its own data tables in the usual fashion.

If one of the new Sheets names an Item Designator already in the data tables, the Base named on the new Sheet will replace the previous Base of the Item, and any new Field Designators will be added to the Item. If the positions of any Fields within the Item are to be changed, the programmer lists those Field Designators, giving their new Locations and partial-word Fields. Thus, the programmer need list on the new Sheets only those Field Designators which are to be added, or whose positions are to be changed.

There is no need to delete Item Designators or Field Designators which are no longer wanted, as they merely remain in the data tables, but are never referred to during the Assembly.

CODING SHEET

Neat CODING SHEET National*

NCR 304 DATA PROCESSOR

PROGRAM ______________________________

REGION ________ WHICH HAS THE FUNCTION ______________________

PROGRAM DESIGNATOR

CODED BY ________________ DATE ________

CHECKED BY ________________ DATE ________

CODE >	Reference No. Region	Reference No. Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&	REMARKS
															>
															>
															>
															>
															>
															>
															>
															>
															>
															>
															>
															>
															>
															>
															>
															>
															>
															>
															>
															>
															>
															>

X-1742-12 5-1-58 The NCR Co.

*Trade-Mark Reg. U. S. Pat. Off.

PAGE NO. ________ WITHIN REGION ________

CODING SHEET

REFERENCE NUMBER:

The **Region** designation may be any two characters, except that the first character must be a letter of the alphabet. Region designations in which the first character is not a letter of the alphabet, are reserved for standard subroutines. Therefore it is impossible for the programmer inadvertently to use any Region designation which is also used by a standard subroutine.

Position must be expressed as three numeric digits.

MONITOR LEVEL:

The column **M** designates the level of automonitoring to which each Instruction in the assembled program will be subject. It has the same significance as **M** in the Processor format of the Instruction.

OPERATION and VARIATION:

In this column is entered any of the 75 Instructions, Pseudo-Instructions, Assembly Instructions, and their variations, listed on page C-1.

If the programmer records a non-existent Operation Code (such as SUBT, instead of SUB, for Subtract), or if the keypunch operator hits a wrong key (punching, for example, AFF instead of ADD), the *Neat* ***ASSEMBLY*** will substitute the "next best" and printout a comment to that effect. When the assembly is complete, the programmer reviews all these comments, and determines what corrections, if any, must be made.

MISCELLANEOUS:

The information to be entered in this column is specifically defined in Appendix C for each Instruction which uses this column.

INDEX REGISTER:

Column R designates the Index Register, if any, to which each Instruction shall be relative. If no Index Register is used with an Instruction, this column is left blank, and keypunched as "zero".

Remember | The program must always contain the Instructions necessary to accomplish all initial presetting of Index Registers.

ADDRESS and TAG COLUMNS:

The possible entries in the address-columns are tabulated below.

There are four possible entries in each of the Tag columns:

Tag "0":	Blank column, keypunched as zero. The entry in the address-column is to be translated into an address or an Address-Type Number.
Tag "C":	The entry in the address-column is a Named Constant of up to 5 alphanumeric characters. See page 9 for use of "+" to reserve left-zeros.

Tag "R": The entry in the address-column is a Constant "once removed".

The *Neat* **ASSEMBLY** must *first* translate it into an address or an Address-Type Number, *then* treat it as a Named Constant.

Tag "X": The corresponding syllable of the assembled Instruction is to be relative to the Index Register which is specified in Column R.

The use of the Tags in the Pseudo-Instruction CNST is slightly different, and is tabulated on page C-10.

The entry in an address-column may be:

• Reference Number	May have Tag "0", "R", "X"
• Item or Field Designator	May have Tag "0", "R", "X"
• 4-digit Main Memory address	May have Tag "0", "R", "X"
• Address of "Special" Cell and partial-word	May have Tag "0", "X"
• Address of Index Register and partial-word	May have Tag "0", "X"
• 4-digit tally or limiter	May have Tag "0", "X"
• Named Constant	Must have Tag "C"

When the Tag is "0" or "X", the entry in the address-column is interpreted by the *Neat* **ASSEMBLY** as follows:

Five characters, leftmost character is @, ¢, □, △, %, £, d, s:

Leftmost 3 characters are address of a "Special" Cell;

Rightmost 2 characters are partial-word.

When used in a full-word operation, such as MOVE, the partial-word designation should be **90.**

Five characters, leftmost character is +:

Leftmost 3 characters specify address of an Index Register;

Rightmost 2 characters are partial-word.

When used in a full-word operation, such as MOVE, partial-word designation should be **90.**

Five characters, leftmost character not one of the above:

Reference Number of an Instruction in the program.

If the programmer names a Reference Number which does not appear in the program, the *Neat* **ASSEMBLY** uses the "next best" and prints a comment.

Four characters or less, one of them a letter of the alphabet:

Item Designator or Field Designator.

If the programmer refers to a Designator which has not been listed on any Data Defining Sheet in the program, the *Neat* **ASSEMBLY** uses the "next best" and prints a comment.

Any Item Designator or Field Designator may be used in a full-word operation, such as MOVE, to refer to the word in which it appears.

Four characters or less, all numeric:

An actual Memory address, a tally or a limiter, expressed as a 4-digit number. The *Neat* **ASSEMBLY** will translate the entry into an Address-Type Number.

The interpretation of File Designator, and other address-column entries in Magnetic Tape Instructions, are discussed on page C-11.

A few Instructions require more than one line in *Neat* **FORMAT**. Only Tag "0" (blank) is permissible in any additional line.

INCREMENTS and DECREMENTS, WITH PARTIAL-WORD:

On pages 6 and 8, the use of an auxiliary line for an Instruction was illustrated. An entry on the auxiliary line contains an increment and a partial-word designation, supplementing the address-reference in the Instruction itself. The increment may be negative, in which case it is called a decrement.

Any Reference Number, Item or Field Designator, or 4-digit Main Memory address may carry such a supplement *unless the entry has Tag "R".* An entry with Tag "R" may have an increment and partial-word, *only* within the Pseudo-Instruction CNST (see page C-10).

The rightmost two digits of the supplement are always interpreted as the *partial-word,* and the leftmost three digits are the *increment* (up to 999) or *decrement* (up to−99). The partial-word designation in a supplement always supplants the normal partial-word associated with an Item or Field Designator; therefore, if an increment or decrement is used, the partial-word designation must always be used with it.

SP& COLUMN:

S in this column indicates that the Sign of V is to be negative.

P in this column Protects the Instruction against the storage of any Named Constants within it. Sometimes a complete Instruction will be replaced by the program itself, and in that case it is essential that no Constants be stored within that Instruction.

& in this column indicates *both* minus-V *and* Protect.

CHANGES TO CODING SHEETS:

Changes in the code, after it has been keypunched, are discussed in the tutorial section of this Manual, pages 10-13.

APPENDIX B

STEP

APPENDIX B

SIGNIFICANT ADDRESSES WITHIN STEP

Address of WRITE-COPY must be stored in 20-field of	@01
First 8 words for WRITE-COPY-READ must be preserved in	¢11-¢18
After end-of-tape intervention, jump to	□18
Sequence Number of next program is stored in 60-field of	¢99
Sequence Number of next overlay is stored in 90-field of	△03
To bring in an overlay, jump with minus-V link to	△98
For next program, jump to	△42

See also pages C-2 and C-3 for conventions governing ¢00-¢09, □00-□09.

USE OF OPTION SWITCHES IN STEP

Option Switch #0 ON:	Override label-check failure.
Option Switch #1 ON:	Operator enters Date This File Recorded before source tape label-check is performed.

STEP

STEP (Standard Tape Executive Program) is a universal subroutine which performs all "housekeeping" functions connected with Magnetic Tape Operations. Using **STEP**, the programmer is never concerned with such conditions as Error or End-of-Tape, and he writes his programs as though these problems did not exist. **STEP** is completely self-contained, and lies entirely within the 400 "Special" Cells which are present with a 2000-word Main Memory; **STEP** occupies no space whatever in Main Memory.

FILE-TABLES:

The *Neat* ***ASSEMBLY*** sets up a series of File-Tables for each program, and these are stored on the Program-Library Tape with the program itself. When the program is being run, the File-Tables govern the action of **STEP** in accommodating all tape-exit conditions to the requirements of each File. The format of a File-Table is:

9	8	7	6	5	4	3	2	1	0
CO$_D$	PH$_D$	AH$_D$	CODE	UNIQUE WORD	INDEX REGISTER		END OF TAPE EXIT		
FILE TITLE (DESTINATION)			CURRENT DESTINATION REEL NUMBER			CURRENT PERIOD NUMBER			
CO$_S$	PH$_S$	AH$_S$	CODE	UNIQUE WORD	0	PERIOD NUMBER WHEN DESTINATION TAPE BECOMES OBSOLETE			
FILE TITLE (SOURCE)			CURRENT SOURCE REEL NUMBER			PERIOD NUMBER WHEN SOURCE TAPE WAS RECORDED			
LENGTH OF SKIP RECORD	NON-EQUAL STOP EXIT			CRM EXIT			BUSY EXIT		

Remember that any body of information recorded on magnetic tape is spoken of as a File. From the point of view of the "housekeeping" operations such as label-checking, there is no distinction between a "file" (in the conventional sense) and any other magnetic tape.

CO$_D$, PH$_D$, AH$_D$ are the Controller, Primary Handler, and Alternate Handler for destination tapes of this File. If there is no destination tape for this File, this field will contain **XXX.**

CO$_S$, PH$_S$, DH$_S$ are the Controller, Primary Handler, and Alternate Handler for source tapes of this File. If there is no source tape for this File, this field will contain **XXX.**

Index Register contains the rightmost two digits of the address of the Index Register used to control alternation of Handlers for this File. If no Index Register has been assigned, this field will contain **10.**

Code represents a bit-wise code in one character:

A	B	C	D	E	F

A: 1-bit if RESCUE POINTS are to be established on this File.

B: 1-bit if automatic SETUP is desired on this File.

C: 0-bit always.

D: 0-bit always.

E: 1-bit if USE LOCKOUT is desired on Rewind.

F: 1-bit always.

Code is set up by the *Neat* ***ASSEMBLY*** in accordance with the entries on the File Specification Sheet, and is identical for both source and destination tapes.

Unique Word is the relative position of the unique word within each record. It is set up by the *Neat* ***ASSEMBLY*** from the entry on the File Specification Sheet, and is identical for both source and destination tapes.

> There may be occasions when the programmer does not wish **Code** or **Unique Word** to be the same for source and destination tapes. He can make these changes in the File-Table after the program has been assembled, and before it is recorded in the Library.

File Titles. The *Neat* ***ASSEMBLY*** enters the File Designator into these fields, since that will usually serve as the File Title for both source and destination tapes.

> These will be occasions (as mentioned on page 23) when different File Titles must be used for source and destination Tape-Labels. The programmer may change either Title in the File-Table after the program has been assembled, and before it is recorded in the Library.
>
> He may also, when required, have the program itself change the Titles in the File-Table, as illustrated on page B-1.

Current Reel Number for source and destination tapes, each start with Reel #1. The appropriate tally is advanced each time an end-of-destination or end-of-source is encountered.

Each **Period Number** is filled in by the daily STARTUP Program, as described on page B-8.

Each of the four **Exits** contains either the address specified on the File Specification Sheet for the respective condition, for this File, or an address within **STEP** for standard handling of the condition.

Length of Skip Record is approximately the same as the longest record in the File. The entry in the File-Table is actually 1/10 of the record-length.

TABULATION OF TAPE-EXIT CONDITIONS:

After any branch out of a Magnetic Tape Instruction, **STEP** examines the 55-field of the second word of that Instruction, to obtain **F**, the File-Table Number corresponding to the File addressed by that Instruction. Stored within this File-Table is information (specified by the programmer on the File Specification Sheet) to enable **STEP** to take the correct action for each condition which may arise in processing that individual File.

BUSY

GOTO the address stored in the 20-field of the 5th word of the File-Table. This is the address specified by the programmer for the BUSY branch.

If the programmer has specified "STICK" on Busy, this is the address of the link within **STEP**, to repeat the Instruction which branched.

USE LOCKOUT

Type LOCKOUT and the contents of Cell @00, then Halt. When the operator presses the START button, repeat the Instruction which branched.

READ ERROR

Try again 3 times. If able to read without error GOTO the next Instruction in the program.

In most Magnetic Tape systems, "reading" errors are usually due to the fact that the information was incorrectly recorded on the tape in the first place. In the 304 System, all recording is automatically verified, *at the time of recording*, and the only reading errors encountered will be true reading failures. Such failures, when they *do* occur, are almost invariably due to a transient condition such as a speck of dust on the tape, and will correct themselves during the 3 re-readings of the record.

In rare cases, the tape may have been worn or aged to the extent that it may accept recording satisfactorily on one day, but some point on the tape may be unable to reproduce that recording on some successive day when the tape is used as a source. In such case, the operator has the option of "jamming" the program past the error, in order to complete the rest of the day's work, and attempt to restore the one or two garbled characters in this record at a later time.

If **STEP** is unable to read any record after a total of four attempts, it will type on the Console Typewriter REC BAD, the garbled record, and the contents of Cell @00. The operator may stop the Processor, go back to the previous period's work to re-create the entire reel of tape, and then resume today's work from the last rescue point. Probably, however, the operator will allow **STEP** to continue, whereupon it will drop the garbled record from the File, and continue the processing. Usually the record can be corrected from other information available, and replaced in the File the following period.

WRITE ERROR

Index the tape backward to the beginning of the Gulp, then rewrite the Gulp, one record at a time (whether they are fixed or variable length records). Ignore end-of-tape warning till Gulp is complete. If unable to write any record without error, try it again.

If still unable to write, assume there is a flaw at that spot, and record a "skip" record. Then resume writing one record at a time till the Gulp is complete. Tally the number of "skip" records on each reel of tape. For detail of "skip" record, see page B-5.

COPY OR COPY-READ ERROR

Try 3 times to resume the operation (Copy or Copy-read portion only).

If unable to resume without error, read one record from the source tape. If this is done without error, the fault must have been on the destination tape, so record a "skip" record. Index Backward the source tape one record, and resume the Copy.

If unable to read, use the "jamming" procedure, which is described under Read Error.

If it was Write-Copy, GOTO the Instruction which branched.

If it was Write-Copy-Read, GOTO the next Instruction.

ERROR IN WRITE PORTION OF WRITE-COPY-READ

The first eight words of the written record, which have been obliterated in Memory, were saved by the programmer in Cells ¢11 thru ¢18. Restore these eight words, and try again 3 times.

If the record still cannot be written without error, record a "skip" record, then try again. Tally the number of "skip" records on each reel of tape.

GOTO the next Instruction.

END OF TAPE

GOTO the address stored in the 20-field of the first word in the File-Table. This is normally □18.

If the programmer wishes to intervene at end-of-tape, this address is the one he specified for his intervention subroutine. This subroutine must end with a jump to Cell □18.

Rewind the tape, and alternate the source or destination Handler-designations in the File-Table and in the appropriate Index Register. Label-check the tape on the alternate Handler.

If end of destination tape, check the label for expiration, Index backward, and record a new label. If rescue points are called for on this File, perform a Memory Dump.

If end of source tape, check the label for File Title, Reel Number, and Date Recorded. If rescue points are called for on this File, Index forward over the Memory Dump which is on the tape.

If end of source tape, type on the Console Typewriter log:

`NEW SRL ON__`(Controller and Handler holding the next reel).

If end of destination tape, type on the Console Typewriter log:

PUT THIS LBL ON__(Controller and Handler holding the rewound reel); then the second word of the File-Table, and the 30-field of the third word of the File-Table, which is the information to be written on the outside of the reel, as a visible label.

If this was a branch without execution . . . .

—Because a previous Write-Copy found end of source tape or end of destination tape, resume the Write-Copy (Copy portion only) and GOTO the Instruction which branched.

If this was a branch after termination . . . .

—Because Write-Copy-Read found end of source tape or end of destination tape, resume the Write-Copy-Read (Copy-Read portion only) and GOTO the next Instruction.

—Because Write Tape found end of destination tape, GOTO the next Instruction.

—Because Read Tape found end of source tape, GOTO the Instruction which branched.

CONTROL RECORD MARK

GOTO the address stored in the 53-field of the fifth word in the File-Table. This is the address specified by the programmer for the CRM branch.

If the programmer entered **N** for this branch on the File Specification Sheet (no CRM expected) this address jumps into **STEP**, which types UNEX CRM and the contents of Cell @00, then halts.

UNEQUAL STOP

This is the one case in which STEP *does not* select the File-Table corresponding to the Instruction which branched. If the Un-Equal Stop terminated an off-line Copy, STEP selects the File-Table corresponding to the Copy Instruction, and then chooses the branch address from that File-Table.

GOTO the address stored in the 86-field of the fifth word of the File-Table. This is the address specified by the programmer for the Un-Equal Stop branch.

If the programmer entered **N** for this branch on the File Specification Sheet (no Un-Equal Stop expected) this address jumps into **STEP**, which types UNEX UES and the contents of Cell @ 00, then halts.

"SKIP" RECORD:

This is a special record, approximately as long as the longest record in the File, containing the configuration SKIP sp v w s d d in its first, fourth and seventh words. It is recorded whenever a persistent Write error indicates that there must be a flaw on the tape. When recording a "skip" record, the Controller will, of course, indicate a Write error; in this case, STEP expects the error, and ignores it.

Whenever a Read error is encountered, STEP examines the first, fourth and seventh words of the error record, since experience indicates that tape flaws, when they *do* occur, are far too small to garble all three of these words. If any one of them has the prescribed configuration, this is a "skip" record; STEP drops it from the File, and resumes the program.

It sometimes happens that an error which persists when attempting to record one configuration of bits will not occur when recording some other configuration. Even if the "skip" record should, by this kind of accident, be recorded on the destination tape without error, **STEP** guarantees that the program will not attempt to treat the "skip" record as though it were a legitimate record in the file.

It will be noticed that the "skip" record is also a CRM record, and will terminate any Read operation. For this reason, before taking any CRM-jump, **STEP** checks the CRM record to see whether it is also a "skip" record; if so, **STEP** drops it from the File, and resumes the program.

On the other hand, the "skip" record is also a Reject record, which can never terminate a Search or Copy operation, regardless of any accidental compliance with the search condition (see pages IV-Tapes-8 and 9 in the 304 Programming Manual). Therefore, if a Copy encounters a "skip" record which was recorded without error, that record will be transcribed harmlessly onto the destination tape. "Skip" records will be rare, and those recorded without error will be rarer still, so this additional accumulation of "skip" records within a File is of no significance.

TALLYING "SKIP" RECORDS:

STEP maintains an independent tally of "skip" records recorded on every Handler in the system. Since this tally is reset to zero every time a new reel of tape is used on the Handler, it is actually a tally of the number of "skip" records on every reel of tape used in the system. Whenever the number of "skip" records on any reel exceeds 30, **STEP** types on the Console Typewriter log SKIP OVER 30 —— naming Controller and Handler, as an indication that it is probably time to discard that tape once its information expires.

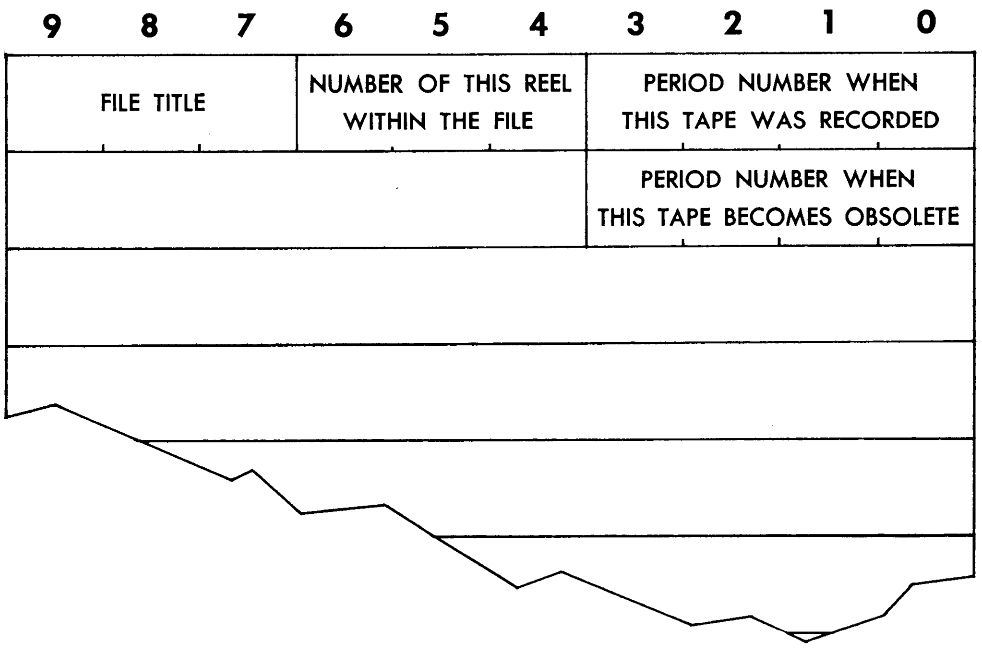
The figure 30 is purely arbitrary, and may be changed at will.

TALLYING ERRORS:

For the benefit of the maintenance personnel, **STEP** maintains an approximate tally of magnetic tape errors in which each Handler is involved.

TAPE LABELS:

The first record on every reel of magnetic tape is recorded as a label, in the following format:


9 8 7 6 5 4 3 2 1 0
FILE TITLE
NUMBER OF THIS REEL WITHIN THE FILE
PERIOD NUMBER WHEN THIS TAPE WAS RECORDED
PERIOD NUMBER WHEN THIS TAPE BECOMES OBSOLETE

The information in the Tape Label identifies every reel, and is used by **STEP** in Label-checking. Only the first word, and the 30-field of the second word, are relevant within the Label; therefore **STEP** records a destination Tape Label as a minimum-length record starting with the second word of the File-Table, and a source Tape Label as a minimum-length record starting with the fourth word of the File-Table.

DAILY STARTUP:

In order to minimize the possibility of manual interference with the 304 System when it is operating, the Console Typewriter is the only device in the System which can be started by the operator. Therefore the day's work is started each morning by means of the Typewriter.

When the day's Input has been placed on the Paper Tape Reader, or the Punched Card Reader, and all Magnetic Tapes have been mounted on their proper Handlers, a short piece of paper tape is placed in the Typewriter's reading station, and the START READ button on the Typewriter is pressed.

The paper tape contains a brief program, which is described in detail in the *Neat* Operating Manual. This program brings into Memory the STARTUP Program, and File-Table #0, from the Program-Library Tape. (In reading the following consult page C-2 in connection with references to the "Special" Cells.) The STARTUP Program brings **STEP** into Memory from the Library, and then it . . . . .

– **Types** `TODAYS PERIOD` **and halts.**

The operator enters Today's Unit Period Number* on the Typewriter keyboard, followed by a "Compute" Code. The STARTUP Program checks to be sure he has made the entry, and puts the information into Cell □00:96. Then it . . . .

– **Types** `CURRENT M-PERIOD` **and halts.**

The operator enters Current Major Period Number*, followed by a "Compute" Code. The STARTUP Program checks to be sure he has made the entry, and puts the information into Cell □01:97. Then it . . . .

– **Types** `TODAYS DATE` **and halts.**

The operator enters Today's Date as a 6-digit number (month, day, year), followed by a "Compute" Code. The STARTUP Program checks to be sure he has made the entry, and puts the information into Cell □00:50, where the main program may find it for editing Output. Then it . . . .

– **Types** `FIRST DO YYYYYYY` **and halts.**

YYY. . . represents the Program Sequence Number* of the program which is usually executed first each day, and is located in Cell ¢99:60. If the operator wishes to execute this program, he strikes "Compute" Code on the Typewriter keyboard. If he wishes to execute some other program as the first one today, he enters its 7-digit Sequence Number on the Typewriter keyboard, and *then* strikes "Compute" Code, in which case the new Sequence Number is automatically stored in Cell ¢99:60.

STEP now takes over. It locates, on the Library Tape, the program whose Sequence Number is in Cell ¢99:60, and reads it into Memory, with its File-Tables. Then it updates each File-Table in turn:

It plants the Current Period Number into the 30-field of the second word of the Table. This is either the Unit Period Number (from □00:96), or **M** followed by the Major Period Number (from □01:97), whichever is appropriate for that File.

When the File-Table first comes into Memory from the Library Tape, the 30-fields of its third and fourth words contain, respectively:

Number of periods, including the processing date, during which destination tape is to be protected (increment), taken from the File Specification Sheet.

Number of periods, prior to the processing date, when source tape should have been recorded (decrement), taken from the File Specification Sheet.

STEP adds the increment to the Current Period Number, to obtain the actual Period Number when today's destination tape will become obsolete, and plants that into the 30-field of the third word of the File-Table. Then **STEP** subtracts the decrement from the Current Period Number, to obtain the actual Period Number when today's source tape should have been recorded, and plants that into the 30-field of the fourth word of the File-Table.

The File-Table now contains the information shown in the illustration on page B-1.

*Each of these terms is defined and discussed later in this Appendix.

If the File is to receive automatic SETUP, **STEP** next performs the operations of SET:D and/or SET:S, so that the main program is releived of these chores.

> In executing SET:D, **STEP** plants **CO$_D$** and **PH$_D$** from the first word of the File-Table, into the 88-field and the 66-field of the designated Index Register to preset the Index Register for the Primary Destination Handler. (If no Index Register is specified for this File, the putaway is made to Cell 0010, where it is ignored). **STEP** then reads the Label-Record from the tape mounted on that Handler, and performs Destination-Tape Label-Check.
>
> In executing SET:S, **STEP** plants **CO$_S$** and **PH$_S$** from the third word of the File-Table, into the 88-field and the 77-field of the designated Index Register to preset the Index Register for the Primary Source Handler. (If no Index Register is specified for this File, the putaway is made to Cell 0010, where it is ignored). **STEP** then reads the Label-Record from the tape mounted on that Handler, and performs Source-Tape Label-Check.

When all the File-Tables have been processed in this manner, **STEP** jumps to the START address of the main program (which was specified on the Program Header Sheet when this program was assembled), and the main program then starts to run.

PROGRAM LABELS:

Each assembled program, when recorded on the Library Tape, is preceded by a Program Label. This Label is a minimum-length record, created by the Librarian Program at the time the assembled program is placed in the Library. The complete contents, and the format, of this Program Label are shown in the *Neat* Operating Manual as part of the discussion of the Librarian Program.

For our present purpose, we note that the information in the Program Label includes:

- Program Sequence Number
- Program Designator
- Program Origin
- START address, if a main program (not an overlay)
- Sequence Number of the program normally executed next.

The first and last of these items are specified at the time the Librarian Program makes up the Label for inclusion in the Library Tape. The rest are specified on the Program Header Sheet at the time of assembly.

Once a program has been assembled, and placed in the Library, the **Program Sequence Number** supersedes the Program Designator as the identification of the program. The Program Designator is not used again unless the program is to be changed and reassembled.

The Sequence Number defines the position of the program within the recording sequence in the Program-Library. It is a 7-digit number for a main program, with the same 7 digits being followed by the 3-digit Overlay Number for Overlays within the program. The Sequence Numbers are completely arbitrary except that, as far as possible, programs should be recorded in the Library in the same sequence as they will most often be used. If some program is used more than once during a Processor run, it might be well to record that program in several places within the Library, using a different Sequence Number each time.

PROGRESSING FROM PROGRAM TO PROGRAM:

As each successive program is read from the Library into the Processor Memory, **STEP** stores the Sequence Number of the *usually-next* program in Cell ¢99:60, where it remains throughout the execution of the present program.

During the present program, some characteristic of the data (or perhaps the state of a Console Option Switch) may effect the choice of the next program to be executed. If so, the present program must plant into Cell ¢99:60 the 7-digit Sequence Number of the program which is *actually* to be next, replacing the previous contents of that Cell.

When the present Program is complete, and it is time to proceed to the next program, the present program must jump to Cell Δ42, within **STEP**.

STEP types out, on the Console Typewriter:

```
PROGRM XXXXXXX COMPLETE

NEXTDO YYYYYYY
```

where XXX . . . represents the Sequence Number of the program just finished, and YYY . . . represents the contents of Cell ¢99:60, the Sequence Number of the next program to be executed. Then **STEP** halts the Processor. If the operator wishes to have some other program executed next, he enters the 7-digit Sequence Number of that program on the Typewriter Keyboard; **STEP** plants this into Cell ¢99:60.

Then the operator strikes "Compute" Code on the Typewriter Keyboard (or presses the START button on the Control Panel); **STEP** finds the next program on the Library Tape, brings it into Memory, updates the File-Tables, executes SET:D and/or SET:S on each File for which automatic SETUP is prescribed, and then jumps to the START address of the program, as described on page B-8.

OVERLAYS:

The procedure for bringing an Overlay program into Memory is similar to that for a main program. The sequence of Overlays usually depends on the data being processed, with Overlays being called in as they are needed; whenever an Overlay is wanted, the main program must plant into Cell Δ03:90 the 10-digit Sequence Number of that Overlay, before calling the Overlay into Memory.

The main program must then jump to Cell Δ98 within **STEP**, with a minus-V link. **STEP** finds the Overlay on the Library Tape, and brings it into Memory.

At this point, the procedure for getting ready to execute an Overlay differs from that for a main program. An Overlay uses the File-Tables of its main program, and has none of its own; therefore **STEP** performs no automatic SET FILE operations for the Overlay. Further, **STEP** does not initiate execution of the Overlay; it returns to the main program, using the minus-V link, and the main program must then jump to the Overlay, as described on page A-8. **STEP** furnishes no type-out when bringing in Overlays; if the programmer wishes to have these events recorded on the Console Typewriter's log, he includes appropriate Instructions in the main program, or in the Overlays.

LABEL-CHECKING

Automatic label-checking of magnetic tapes is a technique which permits the Processor program to verify the work of the *people* at a data processing installation, and to guarantee that they always mount the correct tape on the correct Handler at the correct time. Although every magnetic tape used in the System is clearly labeled on the face of the reel, and although tape-utilization schedules are always carefully maintained, the possibility of human error can never be disregarded. The consequence of such human error is so drastic that the need for the additional protection of programmed label-checking is universally recognized.

It must be admitted that, despite the universal recognition of this need, the use of programmed label-checking is far from universal. The principal objection has been that most label-checking systems are too inflexible for the varied needs of a business data processing system, and they require the exercise of so much human judgment on the part of the Console operator that they offer only illusory protection. In most cases, in order to provide any protection at all, the automatic label-checking system must be *too* conservative; it rejects a large proportion of the tapes used, whereupon the operator must verify whether each reel *is* actually safe to use. If so, he overrides the label-check failure. In practice, these continual interruptions are inconvenient and time-consuming, in addition to the fact that this habitual overriding tends to make the operator careless. It is often felt that under such circumstances an installation is better off without programmed label-checking.

The **STEP** label-checking system has been designed with just these considerations in mind; by setting a new standard of flexibility and control, it interrupts Processor operation only when an improper tape actually has been mounted, and there is no occasion for the Console operator to exercise his discretion at that time.

THE PHILOSOPHY OF LABEL-CHECKING:

The object of label-checking is to guarantee that the correct tape is always being used as a source, and that only an obsolete tape, whose information is no longer of any value, may be used as a destination. For this purpose, the first record on every reel of tape is recorded as a Label-record to identify that reel. As shown on page B-7, the label contains:

File Title;

Number of this reel within the File;

Period Number when this tape was recorded;

Period Number when this tape becomes obsolete.

Because of the variability of processing schedules, and the incidence of weekends and holidays, sequential Period Numbers, rather than calendar dates, are used for dating magnetic tapes. Two independent sets of Period Numbers are offered by **STEP**:

The Unit Period (usually daily), which is a 4-digit number;

The Major Period (usually monthly), which is a 3-digit number.

Almost any common processing cycle can be expressed in terms of one of the other of these Period Numbers. Thus a weekly cycle consists of 5 (or 6) Unit Periods; a quarterly, semi-annual or annual cycle consists of 3, 6 or 12 Major Periods. **STEP** also contains provision for short cycles (due to holidays, for example) and for irregular cycles, such as for reports submitted only on demand; these facilities will be discussed further on.

As described on page B-8, **Today's Unit Period Number**, and the **Current Major Period Number** are always stored in Cells □00:96 and □01:97, respectively. One or the other of these is used to set up in each File-Table the three "dates" which are pertinent to that File. Each of these "dates" will appear as a 4-digit number for a File whose processing cycle is based on Unit Periods, or as **M** followed by a 3-digit number for a File whose processing cycle is based on Major periods. **STEP** performs all label-checking in terms of whichever sequence of Period Numbers is appropriate to the File being checked.

Thus **STEP** furnishes the ability to cope automatically with the two incommensurable sets of processing cycles found in every business data processing system; and in doing so it permits *no manual "tinkering"* with the completely automatic label-checking process.

In the case of a short cycle, or an irregular cycle, the predicted **Period Number when source tape was recorded** (stored in the File-Table) is incorrect, but before starting the day's work the operator knows just when the tape *was* recorded. He turns Console Option Switch #1 ON, and begins the program. **STEP** will then type out the fourth word of each File-Table *before label-checking that File,* and will halt. The fourth word contains **File Title (Source), Current Source Reel Number** (which will always be 1 at this point), and predicted **Period Number when source tape was recorded.** If this File-Table does not pertain to the particular File which is on the irregular cycle, the operator strikes "Compute" Code on the keyboard; **STEP** label-checks that File, then types out the fourth word of the next File-Table and halts again.

When this process reaches the File-Table which contains the incorrect "Date Recorded", the operator simply enters on the Typewriter Keyboard the correct Period Number when this source tape *was* recorded, and then strikes "Compute" Code. **STEP** plants this entry into the File-Table, replacing the Period Number previously stored there, and *then* label-checks this File.

Thus the operator never has the opportunity of saying to himself, "The label-check failed, but I know it's all right, so go ahead." **STEP** forces him to enter the correct "Date Recorded" from his own records, *without permitting him to know what is on the tape label.*

If any label-check *should* fail, **STEP** types a notice on the Console Typewriter, indicating which Controller and Handler holds the incorrect tape, whether it is source or destination, and sufficient information from the File-Table and the Tape Label to show the reason for the label-check failure, as shown on pages B-13 and B-14. **STEP** then halts the Processor. The operator must remove the incorrect tape, mount the correct tape on that Handler, then press the START button on the Console. **STEP** trusts no-one, and immediately proceeds to perform the same label-check on this new tape.

The Program-Library Tape will, of course, have a Label, and the Number of periods during which *this* File is to be protected by expiration-check will be 9999 Unit Periods, giving it permanent protection against accidental recording. When a Library Tape is to be revised, the Librarian Program will record a complete new tape, and then record an expired label on the obsolete one. This accomplishes two desirable things:

- The obsolete Library Tape is now released from protection, and may be used as a destination tape, just like any other expired tape.
- It becomes impossible for someone accidentally to use the expired Library Tape, containing obsolete programs, as a program tape.

This does not forbid retaining both the old and the new version of some of the programs in the Library. They may both be recorded in the revised Library, with different Program Sequence Numbers, and either one remains available to anyone who wishes to use it.

THE MECHANICS OF LABEL-CHECKING:

SOURCE TAPE

If Option Switch #1 is ON, **STEP** types out the fourth word of the File-Table, and halts, When operation is resumed, **STEP** plants any entry made through the Console Typewriter keyboard into the 30-field of the fourth word of the File-Table, and then proceeds to label-check.

STEP reads the first record on the tape into its own Working Storage, and performs a COMPARE EQUALITY between the first word of the label-record, and the fourth word of the File-Table. If these two words are identical, this is the correct reel of tape, and **STEP** returns to the main program.

If these two words are not identical, then at least one of:

File Title

Reel number

Date recorded

does not match, and this is the wrong reel of tape. **STEP** types out on the Console Typewriter, and then halts:

CK STAPE LBL ON__ (Controller and Handler)

FOURTH WORD OF FILE-TABLE ——————— THE TWO WORDS OF TAPE LABEL ———————

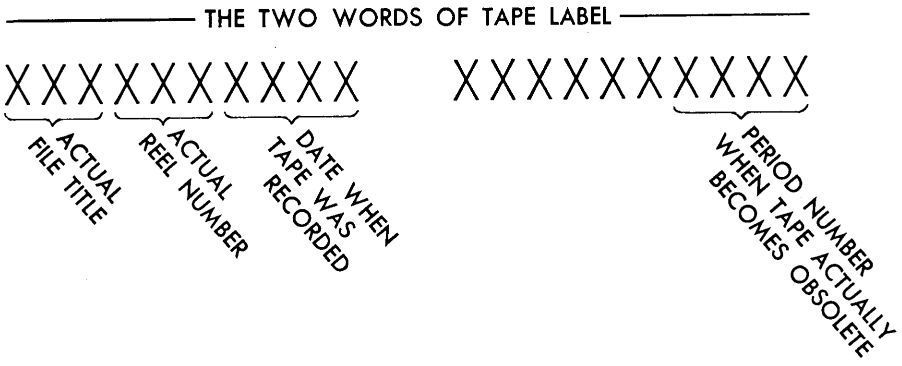
$$\underbrace{\text{XXX}}_{\text{EXPECTED FILE TITLE}}\underbrace{\text{XXX}}_{\text{EXPECTED CURRENT REEL NUMBER}}\underbrace{\text{XXXX}}_{\text{PERIOD NUMBER WHEN TAPE SHOULD HAVE BEEN RECORDED}}\qquad\underbrace{\text{XXX}}_{\text{ACTUAL FILE TITLE}}\underbrace{\text{XXX}}_{\text{ACTUAL REEL NUMBER}}\underbrace{\text{XXXX}}_{\text{PERIOD NUMBER WHEN TAPE ACTUALLY WAS RECORDED}}\qquad\text{XXXXXXXXXX}$$

DESTINATION TAPE

STEP reads the first record on the tape into its own Working Storage, and compares **Period Number when this tape becomes obsolete** with the current Period Number. If the leftmost character in this label-field is **M** the comparison is between the 20-field in the label, and (□01:97). If the leftmost character is numeric, the comparison is between the 30-field in the label, and (□00:96). In either case, if the Period Number in the label is less than, or equal to, the Period Number in Memory, the tape has expired, and may be written on; so **STEP** returns to the main program.

If the Period Number in the label is greater than the Period Number in Memory, the tape has *not* expired, and the label-check fails. **STEP** types out on the Console Typewriter, and then halts:

CK DTAPE LBL ON__(Controller and Handler)


THE TWO WORDS OF TAPE LABEL
XXXXXXXXXX
XXXXXXXXXX
ACTUAL FILE TITLE
ACTUAL REEL NUMBER
DATE WHEN TAPE WAS RECORDED
PERIOD NUMBER WHEN TAPE ACTUALLY BECOMES OBSOLETE

AFTER LABEL-CHECK FAILURE

When operation is resumed after any label-check failure, **STEP** examines Option Switch #0. If this Switch is OFF, **STEP** assumes that the incorrect tape has been manually rewound and changed. It therefore performs the same label-check again on that Handler, as described on page D-x.

If Option Switch #0 is ON, **STEP** types the word OVERRIDE near the right margin of the Console Typewriter log (where it will be quite conspicuous), and returns to the main program as though the label-check has not failed.

OVERRIDING LABEL-CHECK FAILURE:

The success of any label-checking system depends entirely on rigorous control of operator override. The **STEP** label-checking system gives the operator no excuse whatever to override label-checks, except in the most unusual circumstances. Further, the operator and his supervisor should always know in advance when an override will be needed, and the operator should be forbidden to use it without prior authorization. One of the essential audit requirements of the System is a daily review of the Console Typewriter's log, and this review must include checking for unauthorized overrides.

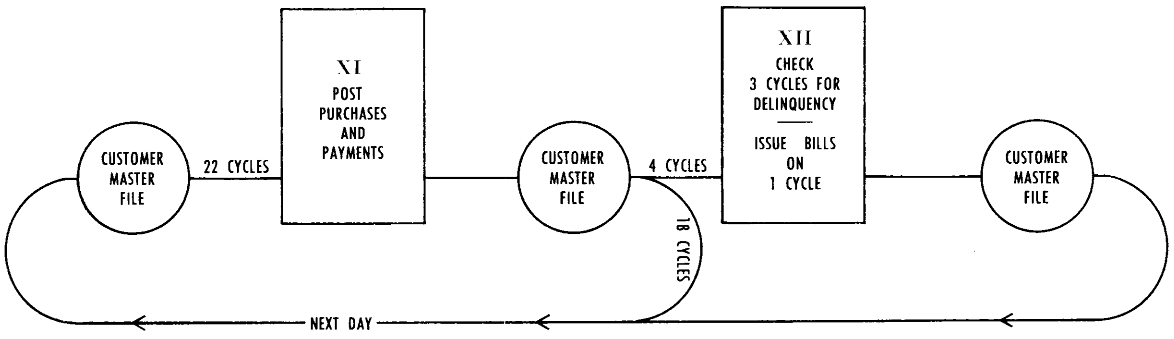
The principal objection to label-checking systems has been that their inflexibility requires frequent operator override of label-check failure. In such systems, the operator must override so often that he tends to become careless, and to override *every* failure without checking it carefully.

STEP, however, furnishes the flexibility and control which are necessary to eliminate the human factor from automatic label-checking. Just as it is the responsibility of the Supervisor of the installation to exercise this control, so it is the responsibility of the programmer to organize the system so that this control can be conveniently exercised. It is well worth a little effort and ingenuity on the programmer's part to adapt **STEP**'s label-checking facilities to his installation's special requirements so as to make it unnecessary, and in fact, impossible, for the Console operator to "tinker" with the system.

The balance of this section deals with some of the "fine points" of label-checking, and shows **STEP**'s flexibility under several conditions in which label-checking is often considered impracticable.

EXAMPLE 1:

This situation arises in a cycle-billing operation. The Customer Master File is divided into 22 cycles, corresponding to the 22 working days in the average month, with customer purchases and payments being posted daily to the entire File. After posting, four of the cycles are selected according to a schedule to go through the next program, where three of them are checked for delinquency, and bills are made up for the fourth. The following illustration shows a portion of the System Model, displaying only these operations on the Customer Master File.

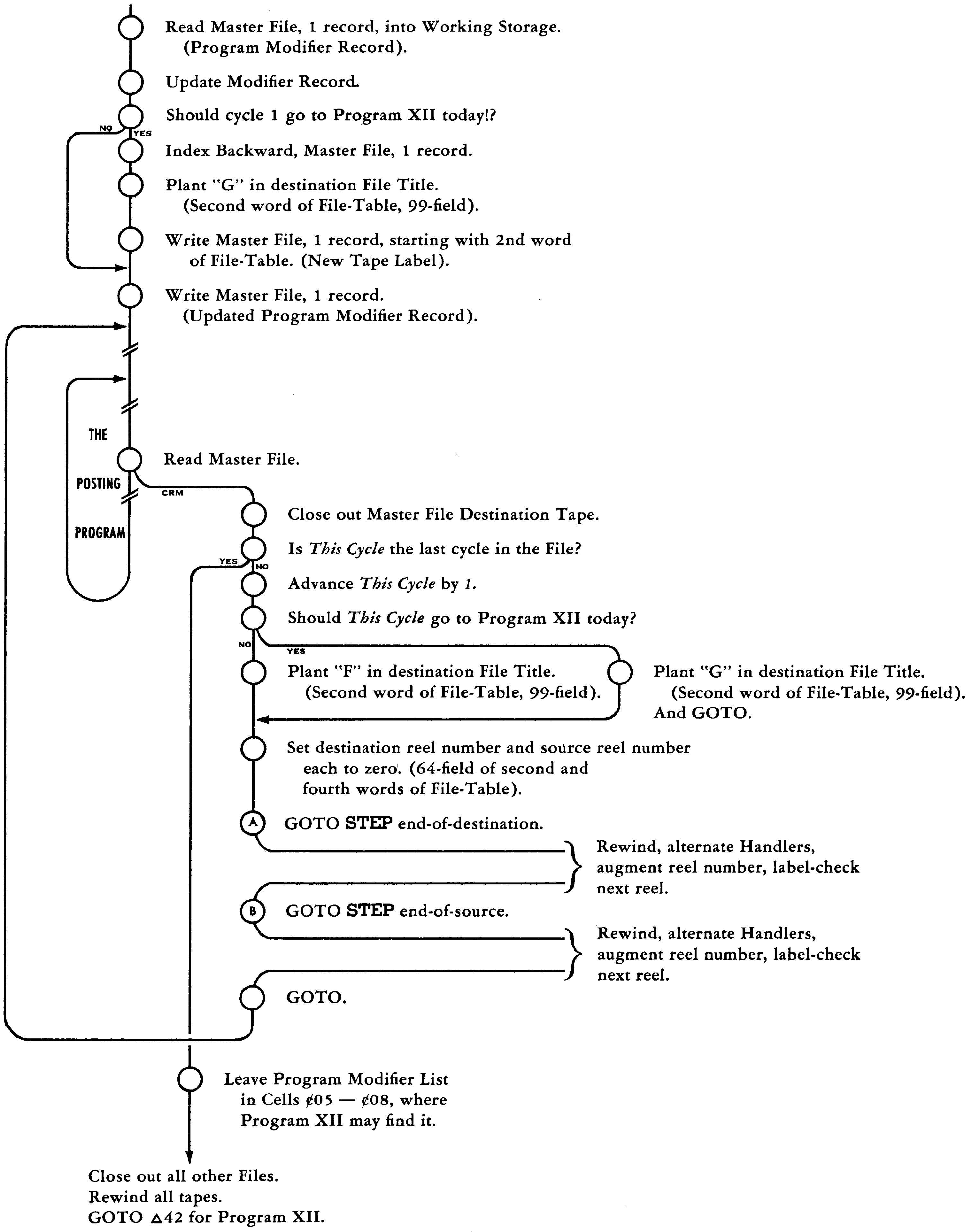


In order to organize this operation conveniently, each of the 22 cycles is set up as a separate File, with its own File Title. Note that the File Titles of the four cycles which are to enter Program XII each day must have their File Titles changed when they come out of Program XI, so that they can never be confused with the tapes which later come out of Program XII.

Set up a system of File Titles in which the first character is either the letter "F" or the letter "G", and the last two characters are the cycle-number. In Program XI, all Master File source tapes contain the letter "F" in their File Titles, and the Master File destination tapes for the 18 cycles which require no further action today will also contain the letter "F"; but the destination tapes for the four cycles which must next enter Program XII will contain the letter "G". In Program XII, all Master File source tapes will contain the letter "G", and all the destination tapes will contain the letter "F".

The first record on the Cycle 1 Master File, immediately following the tape label, is a Program Modifier Record. It contains a list of the four cycles which went through Program XII yesterday, and noting which one of them was billed. The first operation in Program XI, therefore, must update this list, and record the updated Program Modifier Record on the Master File destination tape for Cycle 1. The list is also held in Memory throughout Programs XI and XII, and it governs the File Title assignments, and the label-checking, during both programs.

The File-Table for the Customer Master File in Program XI will initially contain "F01" as File Title for both destination and source tapes. These File Titles will be changed by the program itself as it progresses. The following flow chart shows the operations in Program XI to adapt the File Titles to the **STEP** label-checking system. If different protection intervals are desired for the "F" and the "G" tapes, Program XI may change the Expiration Date in the File-Table at the time it changes the File Title. References to **This Cycle** in the flow chart mean the last two digits of the destination and source File Titles—the 87-field of the second and fourth words of the File-Table. Use automatic SETUP on the Customer Master File in Program XI.


Read Master File, 1 record, into Working Storage. (Program Modifier Record).
Update Modifier Record.
Should cycle 1 go to Program XII today!?
NO
YES
Index Backward, Master File, 1 record.
Plant "G" in destination File Title. (Second word of File-Table, 99-field).
Write Master File, 1 record, starting with 2nd word of File-Table. (New Tape Label).
Write Master File, 1 record. (Updated Program Modifier Record).
THE POSTING PROGRAM
Read Master File.
CRM
Close out Master File Destination Tape.
Is *This Cycle* the last cycle in the File?
YES
NO
Advance *This Cycle* by *1*.
Should *This Cycle* go to Program XII today?
NO
YES
Plant "F" in destination File Title. (Second word of File-Table, 99-field).
Plant "G" in destination File Title. (Second word of File-Table, 99-field). And GOTO.
Set destination reel number and source reel number each to zero. (64-field of second and fourth words of File-Table).
A
GOTO **STEP** end-of-destination.
Rewind, alternate Handlers, augment reel number, label-check next reel.
B
GOTO **STEP** end-of-source.
Rewind, alternate Handlers, augment reel number, label-check next reel.
GOTO.
Leave Program Modifier List in Cells ¢05 — ¢08, where Program XII may find it.
Close out all other Files.
Rewind all tapes.
GOTO ∆42 for Program XII.

Suppose that, within this program, the Reference Number of the Instruction corresponding to operation A is symbolized as RR.PP.0 and the Reference Number of the Instruction corresponding to operation B is symbolized as RR.PQ.0. Then the coding for operation A is:

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
Region	Position												
RR	PP0		DIST				RRPP1	R	¢0990		@0090		
	PP1		CNST		Y		000	C	000	C	□82	C	
	PP2		CNST		0		000	C	F00	C	000	C	

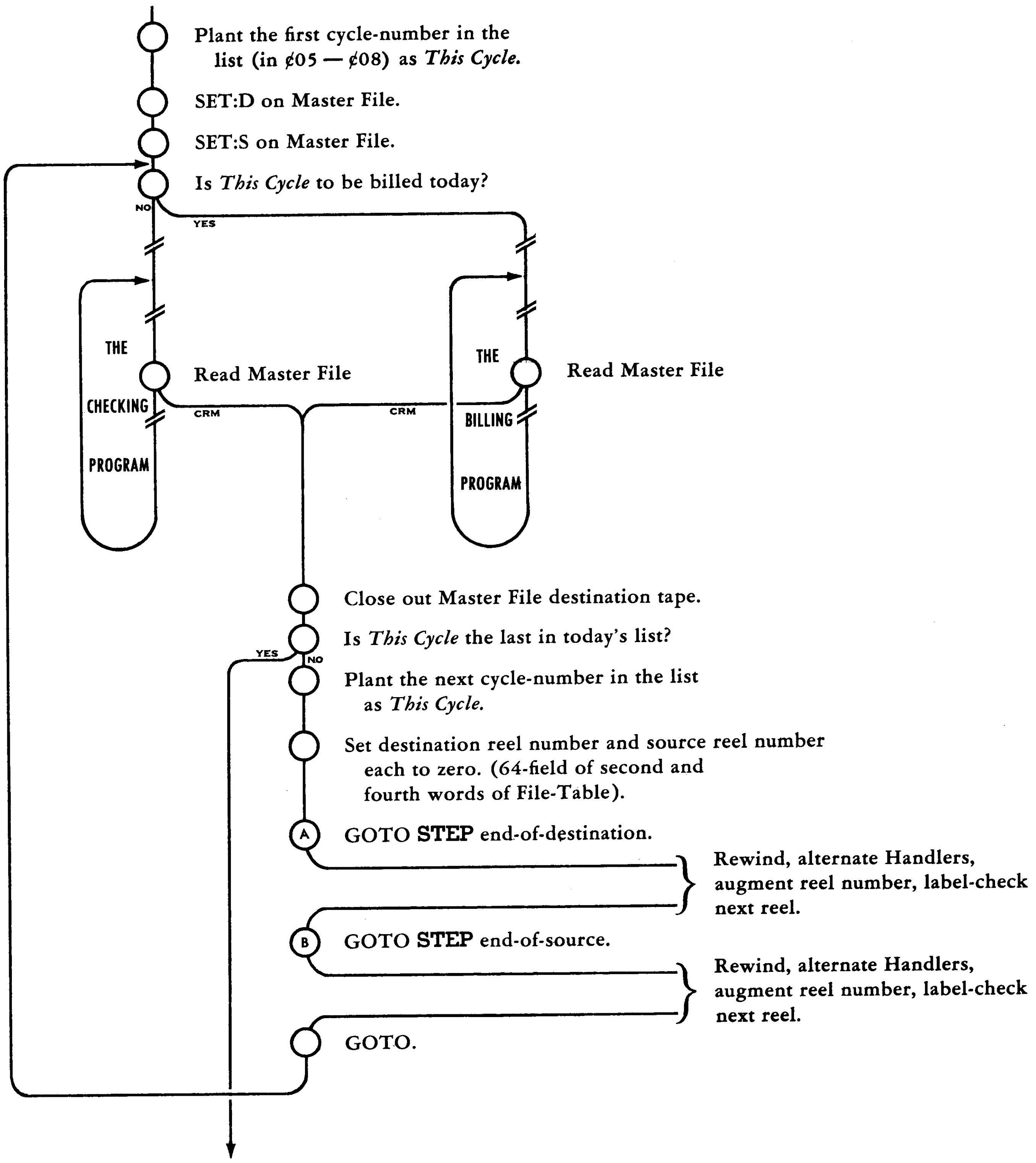
and the coding for operation B is:

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
Region	Position												
RR	PQ0		DIST				RRPQ1	R	¢0990		@0090		
	PQ1		CNST		Y		000	C	000	C	□56	C	
	PQ2		CNST		0		000	C	F00	C	000	C	

The symbox **F** refers to the File-Table Number for the Customer Master File. The same coding applies to these operations on the succeeding flow charts.

The pair of constants in PP.1 and PP.2 are actually an Instruction, written in Processor format within the *Neat* **FORMAT**. They constitute a conventional GOTO Instruction, except that the File-Table Number **F** is in the 55-field of the second word, as though it were a Magnetic Tape Instruction. The same is true of PQ.1 and PQ.2.

The next flow chart shows the operations in Program XII to adapt the File Titles to the **STEP** label-checking system. References to **This Cycle** in the flow chart mean the last two digits of the destination and source File Titles—the 87-field of the second and fourth words of the File-Table. Do not use automatic SETUP on the Customer Master File in Program XII.



Plant the first cycle-number in the list (in ¢05 — ¢08) as *This Cycle.*

SET:D on Master File.

SET:S on Master File.

Is *This Cycle* to be billed today?

Close out Master File destination tape.

Is *This Cycle* the last in today's list?

Plant the next cycle-number in the list as *This Cycle.*

Set destination reel number and source reel number each to zero. (64-field of second and fourth words of File-Table).

GOTO **STEP** end-of-destination.

Rewind, alternate Handlers, augment reel number, label-check next reel.

GOTO **STEP** end-of-source.

Rewind, alternate Handlers, augment reel number, label-check next reel.

GOTO.

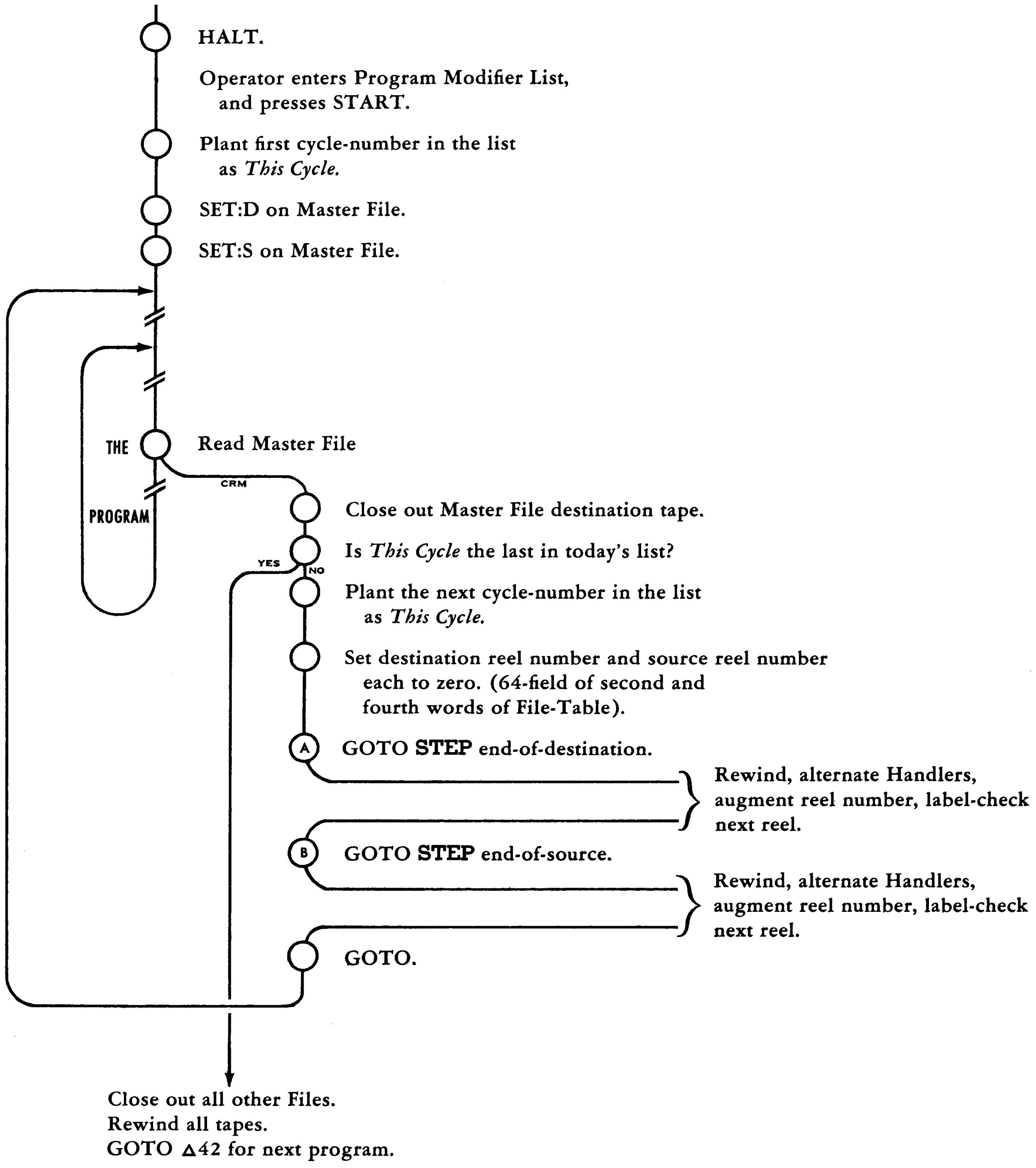
Close out all other Files.
Rewind all tapes.
GOTO Δ42 for next program.

EXAMPLE 2:

In the previous example, the entire file was posted every day, and then those cycles to be processed further each day were selected according to a schedule.

In this example, *all* daily posting is cyclic; there is no single reel of tape which is processed every day, and which could contain a Program Modifier Record. Therefore the Program Modifier List must be entered manually by the Console operator. But note that, even here, he is required to enter this List *before* any label-checking has been performed, so that **STEP** still furnishes an independent verification of the tapes which have been mounted on the Handlers, with no discretion being exercised by the operator.

In this situation, the distinction between "F" and "G" tapes is not needed, and the only modification of File Titles is in **This Cycle**—the 87-field of the second and fourth words of the File-Table. Do not use automatic SETUP on the Customer Master File in this program.



HALT.

Operator enters Program Modifier List, and presses START.

Plant first cycle-number in the list as *This Cycle.*

SET:D on Master File.

SET:S on Master File.

Read Master File

Close out Master File destination tape.

Is *This Cycle* the last in today's list?

Plant the next cycle-number in the list as *This Cycle.*

Set destination reel number and source reel number each to zero. (64-field of second and fourth words of File-Table).

GOTO **STEP** end-of-destination.

Rewind, alternate Handlers, augment reel number, label-check next reel.

GOTO **STEP** end-of-source.

Rewind, alternate Handlers, augment reel number, label-check next reel.

GOTO.

Close out all other Files.
Rewind all tapes.
GOTO Δ42 for next program.

EXAMPLE 3:

In some circumstances, it is necessary to record a Suspense Tape each day, containing items which are not to be processed until the end of the week. These are often items of summary information which will be used for a weekly report. The Suspense Tape, therefore, must build up with the daily recording until the end of the week, when it is used as a source tape for the weekly operation, and then becomes obsolete.

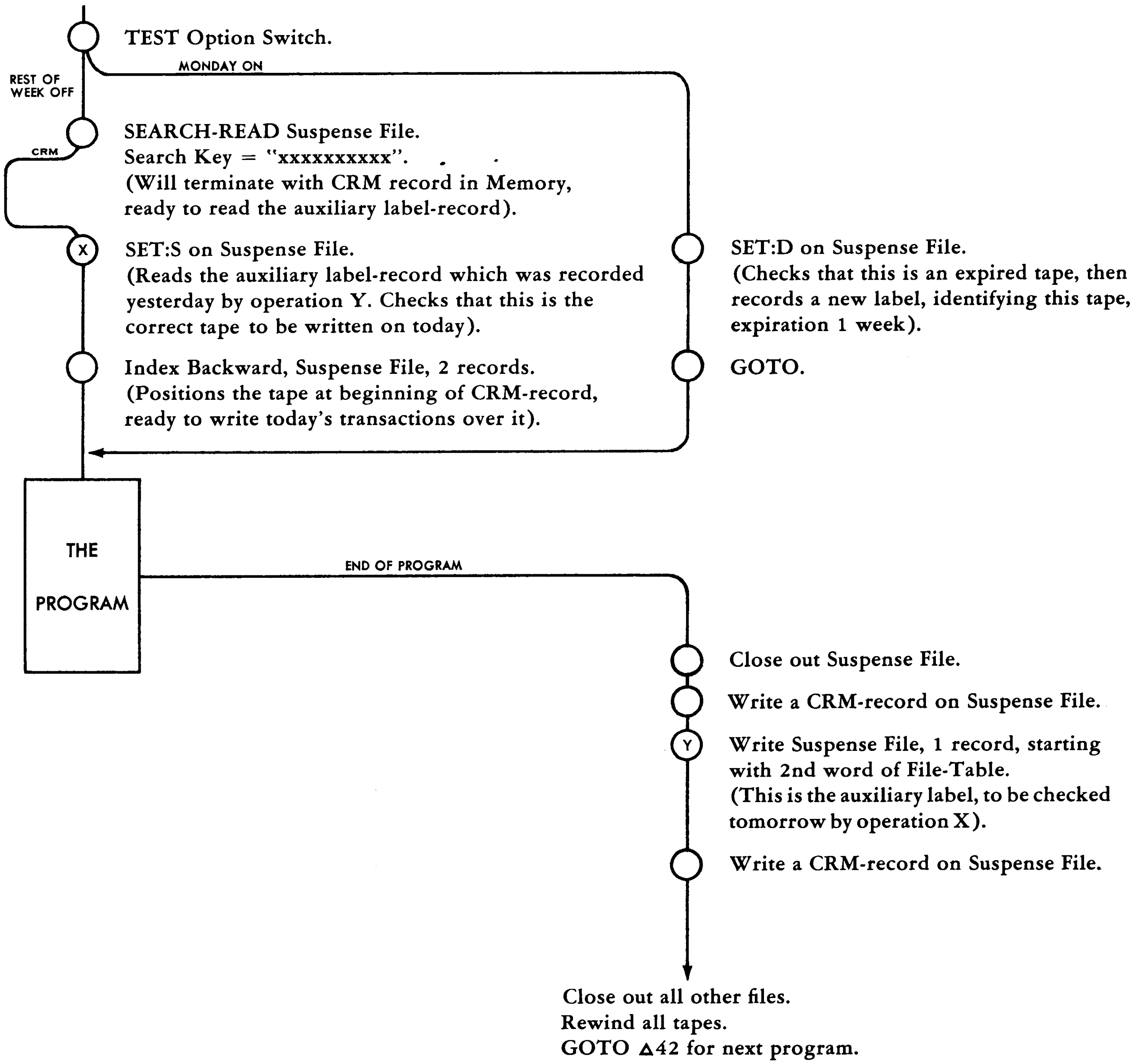
This requirement presents an unusual label-checking condition, since on Monday the normal destination tape label-check is required, to insure that only an expired tape may now be used for recording; but on every other day of the week a kind of source tape label-check must be made to insure that *the correct tape* is about to be written on.

Such a Suspense Tape will contain the conventional label as its first record, with an expiration interval of one week. Monday's program will check this label for expiration, and then record Monday's suspense items. After the last output Gulp has been closed out, and a CRM-record placed on the tape, the program will record an *auxiliary label-record* behind the CRM-record, showing the File Title and Date Recorded (the Expiration Date is immaterial here), followed by another CRM-record.

Every day *except* Monday, the program will initially search for this auxiliary label-record, read it into Memory to verify that this *is* the Suspense Tape which was recorded yesterday, and that this is the point at which yesterday's recording ended. The program then Indexes the tape backward two records to the beginning of the first CRM-record, and is ready to begin recording today's suspense items at that point, obliterating the two CRM-records and the auxiliary label-record between them.

At the end of that day's recording, the program will record a CRM-record, a new auxiliary label-record, and a final CRM-record, to mark the point at which the following day's recording will begin.

The following flow chart shows these operations on the Suspense File. Ordinarily, a Suspense Tape of this sort will not exceed one reel of tape; there will be no alternation of Handlers, no Index Register assigned to this File, and no necessity to preset the Write Tape Instructions for the Suspense File. Do not use automatic SETUP on this File in this program.



TEST Option Switch.

SEARCH-READ Suspense File.
Search Key = "xxxxxxxxxx".
(Will terminate with CRM record in Memory, ready to read the auxiliary label-record).

SET:S on Suspense File.
(Reads the auxiliary label-record which was recorded yesterday by operation Y. Checks that this is the correct tape to be written on today).

Index Backward, Suspense File, 2 records.
(Positions the tape at beginning of CRM-record, ready to write today's transactions over it).

SET:D on Suspense File.
(Checks that this is an expired tape, then records a new label, identifying this tape, expiration 1 week).

GOTO.

Close out Suspense File.

Write a CRM-record on Suspense File.

Write Suspense File, 1 record, starting with 2nd word of File-Table.
(This is the auxiliary label, to be checked tomorrow by operation X).

Write a CRM-record on Suspense File.

Close out all other files.
Rewind all tapes.
GOTO △42 for next program.

APPENDIX C

INSTRUCTION FORMATS

APPENDIX C

The following list gives the OPERATION CODES, OPERATION NAMES, and *Neat* CODES for all operations in the National 304 Data Processor

			Page
1	Add	ADD:	C-4
2	Subtract	SUB:	C-4
3	Multiply	MULT:	C-4
	Round	MULT:R	C-4
4	Divide Right-Justified	DRJ:	C-4
	Round	DRJ:R	C-4
5	Divide Left-Justified	DLJ:	C-4
6	Modify Add	MADD:	C-4
7	Modify Subtract	MSUB:	C-4
8	Extract	EXT:	C-4
9	Insert	SERT:	C-4
□	Add Binary	BINA:	C-4
	Modulo 64	BINA:M	C-4
△	Complement Binary	BINC:	C-4
A	Test Bit	TBIT:	C-4
B	Compare Numeric	CN:	C-4
C	Compare Alphanumeric	CA:	C-4
D	Compare Equality	CE:	C-4
E	Count	CNT:	C-5
F	Test		
	Overflow	TEST:Ø	C-5
	Option Switch	TEST:S	C-5
	Reader Code	TEST:R	C-5
	Punch Code	TEST:P	C-5
H	Combine	CØMB:	C-4
J	Distribute	DIST:	C-4
	Sign Split-off	DIST:S	C-4
K	Suppress	SUPP:	C-4
	Sign Split-off	SUPP:S	C-4
L	Edit	EDIT:	C-4
	Check-Protection	EDIT:P	C-4
M	Merge		
	Cutoff	MRGE:C	C-6
	Runout	MRGE:R	C-6
N	Move	MØVE:	C-6
Ø	Pack	PACK:	C-7
P	Unpack	UNPK:	C-7
Q	Sift	SIFT:	C-7
R	Summarize	SUMM:	C-7

			Page
S	Rewind Magnetic Tape		
	Source Tape	WIND:S	C-16
	Destination Tape	WIND:D	C-16
	Use Lockout, Source	LØCK:S	C-16
	Use Lockout, Destination	LØCK:D	C-16
T	Read Magnetic Tape		
	Complete Records	READ:	C-12
	Partial Records	READ:P	C-12
	Test Branch Conditions	READ:T	C-12
	Index Forward	INDX:F	C-12
	Index Backward	INDX:B	C-12
U	Write-Copy	WC:	C-14
V	Write-Copy-Read	WC:R	C-14
U	Copy	CØPY:	C-15
V	Copy-Read	CØPY:R	C-15
W	Write Magnetic Tape		
	Fixed-Length Records	WT:F	C-13
	Variable-Length Records	WT:V	C-13
	Test Branch Conditions	WT:T	C-13
W	Write Tape and Erase to End		
	Fixed-Length Records	WTE:F	C-13
	Variable-Length Records	WTE:V	C-13
X	Print	PRNT:	C- 8
Y	Type-Punch		
	Console Typewriter (variations 0 thru 5)	TYPE:	C- 8
	High-Speed Punch		
	Fixed Format	PPT:F	C- 8
	Programmed Format	PPT:P	C- 8
Z	Read Paper Tape	RPT:	C- 8
Z	Halt	HALT:	C- 9
*	Read Cards	RCD:	C- 9
PSEUDO-INSTRUCTIONS:			
	Unconditional Jump	GØTØ:	C- 9
	Constant	CNST:	C-10
	Save N Words	SAVE:	C-10
	Set File		
	Source Tape	SET:S	C-16
	Destination Tape	SET:D	C-16
ASSEMBLY INSTRUCTIONS:			
	Omit an Instruction	ØMIT:	C-17
	Next Region	NEXT:	C-17
	Fence	FENC:E	C-18

PROGRAMMING CONVENTIONS

"SPECIAL" CELLS

The conventions and restrictions for use of the "Special" Cells are:

	9	8	7	6	5	4	3	2	1	0
@00	Used by the Processor to store supplemental information generated by Input and Magnetic Tape operations, and by Multiply, Divide, Summarize									

	9	8	7	6	5	4	3	2	1	0
¢00		May be used by the Programmer to store a programmed link			Used by the Processor for self-linking			Used by the Processor as Sequence-Control Register		
¢01	Used by the Processor to store Automonitor information (See Chapter V)									
¢02										
¢03										
¢04										
¢05	To be used by the Programmer as temporary storage for intermediate results of computation or for information which is to be held from program to program during a CONTINUOUS run									
¢06										
¢07										
¢08										
¢09	This Cell will always be named as the irrelevant address whenever only one half of a DISTRIBUTE or a COMBINE is used									

	9	8	7	6	5	4	3	2	1	0
					TODAY'S DATE					
□00	Today's Unit Period Number				Month		Day		Year	
□01	Current Major Period Number			Memory Size as a 4-digit Number				Memory Size as an Address-Type Number		
□02	Additional Fixed Information as required by each installation									
□03										
□04							*	+	−	0
□05	x	x	x	x	x	x	x	x	x	x
□06	C	R	M	space	space	v	space	0	1	0
□07	0	0	0	0	0	0	0	0	0	0
□08	space	space	space	space	space	space	space	space	space	space
□09	9	8	7	6	5	4	3	2	1	0

(□05), (□07), (□08) permit the programmer to pick up variable-length fields of "x's", "zeros", or "spaces" with a minimum of setups.

(□06) is used as the first word of a CRM Record. If the rest of the Record is irrelevant, it may be written directly from this address, as either a Fixed-Length or a Variable-Length Record.

(□09) furnishes a convenient standard location from which to pick up any 1-digit constant. The *Neat* **ASSEMBLY** references all 1-digit Named Constants to Cell □09.

(□04:30) is used in converting a 4-digit Memory address into an Address-Type Number. Assume the address is in Cell XXX:85:

○ **DIST**: (XXX:88)⟶the 55-field and the 44-field of the second word of the next Instruction. (Set up the AL and AR positions.)

○ **SERT**: (□04:--) by (□04:33)⟶XXX:77.

(XXX:75) now contains the Address-Type Number.

THERE SHOULD NEVER BE ANY REASON FOR THE PROGRAMMER TO MAKE A PUTAWAY INTO ONE OF THE □-CELLS.

All other "Special" Cells in a 2400-word Memory are used by STEP, and are not available to the programmer.

The additional 400 "Special" Cells in a 4800-word Memory are available to the programmer without restriction, except that they are sequential only within groups of 200 Cells; there is not, in any useful sense, any "next" Cell after £99 or s99.

INDEX REGISTER #0

Index Register #0 (Cell 000) is reserved for use by STEP, and by other self-contained subroutines. While the programmer is permitted to use Index Register #0 in his own coding, he must remember to preserve its contents before entering any subroutine, and before using any Magnetic Tape Instruction.

CONSOLE OPTION SWITCHES #0 AND #1

These Switches are reserved for use by STEP (Standard Tape Executive Program), and must not be used for any other purpose. They are normally OFF. See Appendix B for description of their use, in connection with Label-Checking of magnetic tapes.

HANDLER #0

Handler #0, on Controller #0, is always reserved for the Program Library Tape.

INSTRUCTION FORMATS

On the following pages are displayed the *Neat* formats, and Processor formats, for all operations. Whenever a *symbol* is shown, it appears in **BOLD** face. Whenever an actual *character* is shown, it appears in `TYPEWRITER` face. No entries are shown for the columns **REFERENCE NUMBER, M, R, TAG, SP&.**

STANDARD OPERATIONS

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
			Op	V			A		B		C		

Op	A		B		C		
V	M	S	R	A_L A_R	B_L B_R	C_L C_R	

1	Add	ADD:	$A + B \longrightarrow C$	
2	Subtract	SUB:	$A - B \longrightarrow C$	
3	Multiply	MULT:	$A \times B \longrightarrow C$	L.J. Next 10 digits $\longrightarrow$ @00:90
	Round	MULT:R		C is rounded
4	Divide Right-Justified	DRJ:	$B \div A \longrightarrow C$	Remainder $\longrightarrow$ @00:90
	Round	DRJ:R		C is rounded. No remainder stored
5	Divide Left-Justified	DLJ:	$B \div A \longrightarrow C$	C is rounded. No remainder stored
6	Modify Add	MADD:	$A \oplus B \longrightarrow C$	
7	Modify Subtract	MSUB:	$A \ominus B \longrightarrow C$	
8	Extract	EXT:	A by B $\longrightarrow$ C	C is automatically cleared first
9	Insert	SERT:	A by B $\longrightarrow$ C	
□	Add Binary	BINA:	$A \perp B \longrightarrow C$	
	Modulo 64	BINA:M		No carry between characters
Δ	Complement Binary	BINC:	A comp $\longrightarrow$ C	
H	Combine	CØMB:	A & B $\longrightarrow$ C	
J	Distribute	DIST:	A $\longrightarrow$ B & C	C is R.J; then B is L.J.
	Sign Split-off	DIST:S		
K	Suppress	SUPP:	A $\longrightarrow$ B & C	B and C are zero-suppressed
	Sign Split-off	SUPP:S		
L	Edit	EDIT:	A $\longrightarrow$ B & C	B is zero-suppressed
	Check-Protection	EDIT:P		

STANDARD COMPARISONS

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
			Op				A		B		J		

Op	A		B		J		
0	M	S	R	A_L A_R	B_L B_R	▨	

A	Test Bit:	TBIT:	If $\left\{\begin{matrix}\text{A has all 1-bits of B}\\ \text{B has all 0-bits of A}\end{matrix}\right\}$	jump to J
B	Compare Numeric	CN:	If A is greater than B	jump to J
C	Compare Alphanumeric	CA:	If A is greater than B	jump to J
D	Compare Equality	CE:	If A is identical with B	jump to J

COUNT

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			CNT		$I\ I_R I_T$		A		T		J		

E	A			T		J		
0	M	S	R	$A_L A_R$	0	I_R	I_{TR}	I

1) MADD: $(00I{:}9I_R) \oplus A \longrightarrow 00I{:}9I_R$

2) If T does not equal contents of the tested syllable of (00I), jump to J.
I_T designates which syllable is to be tested.

I_T may be written as: 9, A, B, C.

TEST OVERFLOW

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			TEST	O							J		

F	▨	▨	▨	▨	J
0	M	S	R	▨	▨

If previous Instruction set Overflow Alarm, jump to J.

TEST OPTION SWITCH

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			TEST	S			N				J		

F	▨	▨	N	▨	J
2	M	S	R	▨	▨

If Console Option Switch #N is turned on, jump to J.
N may be 0, 1, . . . 9.

TEST READER CODE

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			TEST	R			N				J		

F	▨	▨	N	▨	J
4	M	S	R	▨	▨

If High-Speed Paper Tape Reader is set to read Code #N, jump to J.
N may be 0, 1, 2.

TEST PUNCH CODE

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			TEST	P			N				J		

F	▨	▨	N	▨	J
6	M	S	R	▨	▨

If High-Speed Paper Tape Punch is set to Punch Code #N, jump to J.
N may be 0, 1.

MERGE—Cutoff

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
			MRGE	C	L		A		B		C		
			1		K		N_A		N_B		N_C		
			2						X: X_L X_R		Y: Y_L Y_R		
			3				JA		JB		JC		

M	A	B	C
V	M S R	X_L X_R	Y_L Y_R ▨
0	AF	BF	CF
0	0 0 0	0 0 0	0 0 0
▨	▨	L X	Y
0	JA	JB	JC

MERGE—Runout

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
			MRGE	R	L		A		B		C		
			1		K		N_A		N_B				
			2						X: X_L X_R		Y: Y_L Y_R		

M	A	B	C
V	M S R	X_L X_R	Y_L Y_R ▨
0	AF	BF	▨
0	0 0 0	0 0 0	0 0 0
▨	▨	L X	Y
▨	▨	▨	▨

A, ƀ, C:	Designate first word, first item, each string.
N_A, N_B, N_C:	Number of items in each string.
X: X_L X_R:	Relative position of Major Key (if any), within item.
Y: Y_L Y_R:	Relative position of Minor Key within item.
K:	Number of keys (1 or 2) for the Merge.
L:	Length of each item.
JA, JB, JC:	Specify the three exits of Cutoff Merge.

MOVE

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
			MØVE				A		N		C		

N	A	N	C
0	M S R	▨	▨

Transcribe the N words starting at A, into N Cells starting at C.

PACK

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			PACK				A		N		C		
			1								J		

θ	A			N	C
0	M	S	R	▨	J

Pack N word-triples starting at A, into N word-pairs starting at C.
If any zone bits in the data are 1-bits, jump to J, after Packing is finished.

UNPACK

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			UNPK				A		N		C		

P	A			N	C
0	M	S	R	▨	▨

Unpack N word-pairs starting at A, into N word-triples starting at C.

SIFT

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			SIFT		L		A		B		C		
			1		K		N		X: $X_L X_R$		Y: $Y_L Y_R$		

Q	A			B		C
V	M	S	R	$X_L X_R$	$Y_L Y_R$	▨
0	AF			L	X	Y

A:	Designates first word, first item in the list.
B:	Designates first word of the sieve-item.
C:	Designates the location in which the tallies are to be stored.
N:	Number of items in the list.
X: $X_L X_R$:	Relative position of Major Key (if any), within the item.
Y: $Y_L Y_R$:	Relative position of Minor Key, within the item.
K:	Number of keys (1 or 2) for the Sift.
L:	Length of each item.

0	# Words	# Words	# Items

Number of items (and number of words in those items) whose keys are less than, or equal to, the key of the sieve.

SUMMARIZE

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			SUMM		L		A		B		C		
			1				JC		J@				

R	A			B		C
0	M	S	R	$A_L A_R$	$B_L B_R$	$C_L C_R$
0	JC			J@		L

A:	Designates the first field included in the Summary.
B:	Designates the field containing the Number of Items to be Summarized.
C:	Designates the putaway field for the Summary.
L:	Length of each item.
JC:	If the C-putaway overflows, jump to JC, after the Summary is complete.
J@:	If (@00:90) also overflows, jump to J@, after the Summary is complete.

PRINT

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
			PRNT				A				J		

X	A			▨	J
0	M	S	R	▨	▨

A: Designates slew-control word. The 12 words immediately following the slew-control word, constitute the edited print-line.

J: If, after the previous print-line, a "skip" code was encountered on the Printer's slew-control loop, jump to J instead of executing this PRNT Instruction.

TYPE on Console Typewriter

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
			TYPE	V	G		A		N		J		

Y	A			N		J
V	M	S	R	AL AR	0	G

PUNCH PAPER TAPE—High-Speed Punch

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
			PPT	V	G		A		N		J		

A: Designates first field to be output.
N: Number of fields to be output.
G: Output a field from every G^th^ cell.
J: Unconditionally jump to J, after output.
V:

	PROGRAMMED FORMAT	FIXED FORMAT	
TYPE	5	4	Type Only
TYPE	3	2	Punch Only
TYPE	1	0	Type & Punch
PPT	P	F	

READ PAPER TAPE

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
			RPT				A		N		J		

Z	A			N		J
6	M	S	R	AL AR	▨	▨

A: Designates first putaway field.
N: Make not more than N putaways.
J: Designates location of Jump-Table:

	J3	J2	J1

J1: Terminate after N putaways.
J2: End of Tape.
J3: Parity Error in the Tape.

Processor takes next Instruction in sequence, if termination on Compute Code.

After termination, (@00:86) contains Number of Putaways made.
(@00:33) contains Number of characters in last Putaway.

READ PUNCHED CARDS

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
			RCD		W		A		N		J		

*	A			N	J
0	M	S	R	▨	W

Read first W words from each of N cards.
Store first word in Cell A.
If Hopper empty before N cards, <u>and</u> if Last Batch, jump to J.

After termination, (@00:86) contains Number of Putaways made.

HALT

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
			HALT				A				J		

Z	A			▨	J
0	M	S	R	▨	

After the Halt:

A: Any input from Console Typewriter is putaway into Cell A and successive Cells.

J: Unconditionally jump to J, when Console START button is pressed.

When operation is resumed, (@00:86) contains Number of Putaways made.
(@00:33) contains Number of characters in last Putaway.

GO TO

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
			GØTØ								J		

Y	▨			0 0 0	J
0	M	S	R	▨	

This "Instruction" is used as a convenient way of writing an unconditional Jump, and makes it easy to identify the Jump when reviewing the program.

GØTØ is translated into TYPE zero words (and Jump).

CONSTANT

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
			CNST										

9	8 7 6	5 4 3	2 1 0

CNST becomes a word of Constants, at the designated position within the region. Any Instruction in the program will refer to a field within this word by its Reference Number (Region and Position), and by partial-word selectors.

MISC: The right-most character appearing in this column will occupy the 99-field of the word.

A, B, C: The contents of each of these columns will, after translation, occupy the 86-field, the 53-field, and the 20-field, respectively, of the word. The nature of the translation is determined by the Tag of each column.

Tag "0": Zero is not admissable as a Tag in CNST.

Tag "C": The right-most 3 characters in the column will occupy the corresponding 3-character field of the word. Since there is now no problem of reserving significant zeros on the left, the character "+" will always remain a "+".

A reference to one of the "Special" Cells must always be made by its 3-character address, with Tag "C".

Tag "R": As elsewhere, this Tag indicates a Constant "once removed"; that is, a Designator which must be translated into an Address or an Address-Type Number, and which then becomes a Constant. This translated Designator will occupy the corresponding 3-character field of the word of Constants. The Designator may be:

A reference Number within the program;

An Item Designator, or a Field Designator;

A Memory Address, expressed as a 4-digit number.

Within a CNST is the only place where a designator with Tag "R" may be associated with an increment (or decrement).

Tag "X": X is not admissable as a Tag in CNST.

SAVE

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
			SAVE		N		ZERØ						

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	S P &
			SAVE		N		SPACE						

This "Instruction" causes *Neat* to reserve N words at the designated position in the region, and to fill those words with either zeros or spaces, as specified.

MAGNETIC TAPE INSTRUCTIONS

Every assembled Magnetic Tape Instruction contains, in the 55-field of its second word, the File-Table No. **F** assigned to the File addressed by that Instruction. If a Magnetic Tape Instruction is ever used to address different Files alternatively (as in file-splitting on two or more Controllers), it is the programmer's responsibility to plant the alternative File-Table Number into the Instruction whenever its function changes.

The several ways of designating the File in a Magnetic Tape Instruction, and the assembled results corresponding to them, are shown in the table:

Entry in A-column on Coding Sheet	Does the File Spec Sheet for this File Specify an Index Register?	Contents of A-syllable in Assembled Instruction	File-Table No. and Index Register
File Designator	Yes	**0 0 0** A-syllable is automatically relative.	**F** and **R** (obtained from the File-Table for this File Designator) are inserted into the assembled Instruction.
File Designator	No	**Co Sh Dh** A-syllable is relative only if A-column has Tag "X".	**F** (obtained from the File-Table for this File Designator) is inserted in the assembled Instruction. The entry in the R-column of the Coding Sheet is inserted as **R** in the assembled Instruction.
F Co Sh Dh	Irrelevant	**Co Sh Dh** A-syllable is relative only if A-column has Tag "X".	**F** (obtained from the A-column of the Coding Sheet) is inserted in the assembled Instruction. The entry in the R-column of the Coding Sheet is inserted as **R** in the assembled Instruction.

The C-address of every Magnetic Tape Instruction is @02. This is the address of the first word of the Jump Table which is used by every Magnetic Tape Instruction in the program.

READ MAGNETIC TAPE—Complete Records

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			READ		N		FILE		B		Z		

T	FILE			B		@ 0 2
0	M	S	R	F	N	Z

READ from the Source-Tape of the designated File, N records, but not exceeding Z words of Memory.

Store the information in Memory, starting at Cell B.

READ MAGNETIC TAPE—Partial Records

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			READ	P	N		FILE		B		W		

T	FILE			B		@ 0 2
2	M	S	R	F	N	W

READ from the Source-Tape of the designated File, the first W words from each of the next N records.

Store the information in Memory, starting at Cell B.

READ—TEST

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			READ	T			FILE						

T	FILE			▨		@ 0 2
2	M	S	R	F	0 0	▨

Do not Read. TEST branch conditions only.

INDEX FORWARD

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			INDX	F	N		FILE						

T	FILE			▨		@ 0 2
K	M	S	R	F	N	0 0 0

The Variation Designator is K rather than 2 in order that STEP may discriminate more easily between READ PARTIAL and INDEX FORWARD.

INDEX BACKWARD

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			INDX	B	N		FILE						

T	FILE			▨		@ 0 2
4	M	S	R	F	N	▨

Move the Tape forward or backward N records without reading.

WRITE MAGNETIC TAPE—Fixed Length Records

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
Region	Position												
			WT	F	N		FILE		B		L		

W	FILE			B		@ 0 2
0	M	S	R	F	N	L

WRITE on the Destination-Tape of the designated File, N records, each L words long.

Obtain the information from Memory, starting at Cell B.

WRITE MAGNETIC TAPE—Variable Length Records

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
Region	Position												
			WT	V	N		FILE		B				

W	FILE			B		@ 0 2
2	M	S	R	F	N	▨

WRITE on the Destination-Tape of the designated File, N records. The length of each record is specified in the 20-field of the first word of that record.

Obtain the information from Memory, starting at Cell B.

WRITE TAPE & ERASE—Fixed Length Records

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
Region	Position												
			WTE	F	N		FILE		B		L		

W	FILE			B		@ 0 2
1	M	S	R	F	N	L

WRITE TAPE & ERASE—Variable Length Records

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
Region	Position												
			WTE	V	N		FILE		B				

W	FILE			B		@ 0 2
3	M	S	R	F	N	▨

WRITE on the Destination-Tape of the designated File, N records (Fixed or Variable Length). Then ERASE the tape to the end of the reel.

Ignore End-of-Tape Warning.

WRITE—TEST

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
Region	Position												
			WT	T			FILE						

W	FILE			▨		@ 0 2
0	M	S	R	F	0 0	▨

Do not Write. TEST branch conditions only.

WRITE—COPY

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
Region	Position												
			WC		KIND		FILE		B		L		
			1		D Y								

U	FILE		B			@ 0 2	
V	M	S	R	F	D	Y	L

WRITE—COPY—READ

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
Region	Position												
			WC	R	KIND		FILE		B		L		
			1		D Y								

V	FILE		B			@ 0 2	
V	M	S	R	F	D	Y	L

WRITE a single L-word record on the Destination-Tape of the designated File, obtaining the information from Memory starting at Cell B. Then COPY from the Source-Tape to the Destination-Tape, using the contents of Cell BΘ1 as the Search Control.

Y: Relative position (0 to 7), within each record, of the word containing the Search Key.

D: Divides the Search Control into left-hand (LH) and right-hand (RH) portions. D is the number of characters (0 to 7) in the RH portion.

KIND: Kind of Search desired.

Search is performed on the LH and RH portions independently, and each portion may be tested either for Equality (E) or for Range (R). Further, the operation can be made to terminate when either portion of the Search is satisfied (LH or RH), or when both portions are satisfied (LH and RH).

OR		AND
EΘE	LH Equality, RH Equality	EAE
EΘR	LH Equality, RH Range	EAR
RΘE	LH Range, RH Equality	RAE
RΘR	LH Range, RH Range	RAR

If D = 0 (Search Control not divided) then KIND will be stated merely as "E" or "R".

WC: WRITE-COPY (off-line Copy)

Remember | The programmer must see to it that the address of the WRITE-COPY Instruction appears in (@01:20) at the time the next Tape Instruction on this Controller is executed.

WC:R WRITE-COPY-READ (on-line Copy)

Remember | Before executing the WRITE-COPY-READ Instruction the first 8 words of the written record must be preserved in Cells ¢11 thru ¢18.

COPY

Reference No. Region	Reference No. Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			CØPY		KIND		FILE		B				
			1		D Y								

U	FILE			B			@ 0 2
V	M	S	R	F	D	Y	0 0 0

COPY—READ

Reference No. Region	Reference No. Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			CØPY	R	KIND		FILE		B				
			1		D Y								

V	FILE			B			@ 0 2
V	M	S	R	F	D	Y	0 0 0

Same as WRITE-COPY and WRITE-COPY-READ, except that the WRITE portion is omitted.

CØPY: COPY (off-line Copy)

Remember | The programmer must see to it that the address of the COPY Instruction appears in (@01:20) at the time the next Tape Instruction on this Controller is executed.

CØPY:R COPY-READ (on-line Copy)

SEARCH (Off-line Search)

SEARCH-READ (On-line Search)

No *Neat* format is provided for these operations. The programmer must write these Instructions as either CØPY or CØPY:R, in such a way that the Destination Handler is specified as 8 or 9.

If an Index Register is specified for this File, the Instruction should name the FILE in its A-syllable in the conventional way, and the Instruction should be preceded by:

Reference No. Region	Reference No. Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
			SUB				8	C	+0R66		X		
											66		

Where "X" represents the Reference Number of the SEARCH or SEARCH-READ Instruction.

If no Index Register is specified, the Instruction must name the FILE in its A-Syllable as F Co Sh 8.

Remember | The programmer must see to it that the address of the Off-line SEARCH Instruction appears in (@01:20) at the time the next Tape Instruction on this Controller is executed.

REWIND

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
Region	Position												
			WIND	V			FILE						

S	FILE		▨	@ 0 2	
V	M	S	R	F	▨

REWIND WITH USE LOCKOUT

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
Region	Position												
			LØCK	V			FILE						

WIND:S	Rewind Source-Tape without Use Lockout.
WIND:D	Rewind Destination-Tape without Use Lockout.
LØCK:S	Rewind Source-Tape and set Use Lockout.
LØCK:D	Rewind Destination-Tape and set Use Lockout.

SET FILE—Source Tape

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
Region	Position												
			SET	S			FILE						

X	J	X⊕3			□	1	7	@	0	0
X⊕1	0	0	0	0	4	0	7	7	9	0
X⊕2	Y	▨			0	0	0	□	5	6
X⊕3	0	0	0	0	F	0	0	X⊕2		

SET FILE—Destination Tape

Reference No.		M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
Region	Position												
			SET	D			FILE						

X	J	X⊕3			□	1	7	@	0	0
X⊕1	0	0	0	0	4	0	7	7	9	0
X⊕2	Y	▨			0	0	0	□	8	2
X⊕3	0	0	0	0	F	0	0	X⊕2		

SET:S sets the Index Register (if any) for Controller and Primary Source Handler; then label-checks the first reel of Source Tape.

SET:D sets the Index Register (if any) for Controller and Primary Destination Handler; then label-checks the first reel of Destination Tape, and records a new label.

The "Instruction" SET is translated into two actual Instructions:
DISTRIBUTE, which makes appropriate pre-sets in STEP.
GØTØ an address within STEP.

Remember There is a question on the File Specification Sheet "Is automatic SETUP desired on this File?" Whenever this question has been answered "Yes" STEP will automatically SET:S and/or SET:D for that File before allowing the main program to begin, and the programmer will have no occasion to use SET in his program.

Under some circumstances (for example, the File in question may not be available at the beginning of the program), automatic SET FILE will not be used, and the programmer himself will insert SET:S and/or SET:D into his program, at the proper point.

ASSEMBLY INSTRUCTIONS

The programmer may give *Neat* certain Instructions, which will govern the assembly process, but will insert nothing into the program. These ASSEMBLY INSTRUCTIONS are written just like ordinary Instructions, with appropriate Reference Numbers, but *Neat* interprets them as directions to itself.

OMIT

Reference No.														
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
R	P		ΘMIT										

RP: Reference Number (Region and Position) of the Instruction to be omitted.

Neat will omit from the assembled program, the Instruction which has the same Reference Number as the ΘMIT, provided the ΘMIT appears later than the Instruction which is to be omitted.

NEXT

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
	99 8		NEXT		NEXT REGION								

Neat normally assembles the various Regions of a program in the same sequence as they are presented to it. However, the "Instruction" NEXT causes the designated Region to be next in the assembled program, regardless of the sequence of Regions in the program as written.

NEXT permits the programmer to assure that, whenever the sequence of Regions is important, *Neat* will give him the desired sequence.

Remember | The "Instruction" NEXT must always be the last (except for FENCE) of the Region in which it appears, and its Reference Number must designate Position 99.8 in that Region.

FENCE

Reference No.													
Region	Position	M	Operation	V	Misc.	R	A	Tag	B	Tag	C	Tag	SP&
	99 9		FENC	E									

Neat normally stores Constants anywhere in the program, in the irrelevant fields of the Instructions. However, if a portion of the program is to be covered by an Overlay Program, it is essential that no Constants pertaining to any other portion of the program be stored within the Regions which are to be overlaid.

In such a case, the programmer must segregate the Overlay Area, by specifying FENCES to mark its boundaries. The FENCED area extends from the end of one Region (the one immediately preceding the Overlay Area) to the end of another Region (the last one in the Overlay Area).

Neat will never cross a FENCE to store a Constant. That is, all Constants will be stored on the same side of a FENCE as the Instructions which refer to them.

Remember The "Instruction" FENCE must always be the last of the Region in which it appears, and its Reference Number must designate Position 99.9 in that Region.

Remember The execution sequence of the Instructions in a program cannot cross a FENCE except by means of a GØTØ.