NCR

REFERENCE MANUAL

NEAT

COMPILER

•

NCR 315

315

NEAT COMPILER*

THE NATIONAL CASH REGISTER COMPANY

DAYTON 9, OHIO, U.S.A.

PRINTED IN U. S. A.

The NCR 315 NEAT (National Electronic Autocoding Technique) System is a series of programs and techniques developed by the National Cash Register Company for the continued superior support of NCR computer products and customers. This is a complete package to enable the programmers to write and process efficient computer programs with a minimum expenditure of time and effort. All of the programs which make up the series will run efficiently in conjunction with one another.

In order to obtain the greatest benefit from the NCR 315 NEAT Compiler, it is necessary to understand the functions of the associated programs (CRMX-II and the CRAM Librarian or STEP and the Magnetic Tape Librarian). Where necessary to the understanding of the Compiler, details of these programs have been included in this manual; however, for complete information about the supporting programs in the NEAT System, refer to the manual covering that particular program.

TABLE OF CONTENTS

I. INTRODUCTION 1

- A. Equipment Requirement 1
- B. Compiler Functions 1
  - 1. Actual Machine Code 1
  - 2. Automatic Programming 1
    - a. Symbolic coding 2
    - b. Assignment of absolute memory addresses 2
    - c. Assembly function 2
    - d. Printout 3
    - e. Remarks 3
    - f. Control instructions 3
  - 3. Macro Instructions 4
  - 4. Scientific Subroutines 5
- C. File Tables 5
- D. Executive Routines 5
- E. Librarian 5
- F. Procedure 5
- G. Sequence of Events 6

II. EXPLANATION OF THE PROGRAMMING WORKSHEET 8

- A. Programming Worksheet 8
  - 1. Page and Line 8
  - 2. Reference 10
  - 3. Operation (Op:V) 10
    - a. 315 machine instructions 11
    - b. Control instructions 11
    - c. Macro instructions 11
  - 4. Length 11
  - 5. X 12
  - 6. Operands and Remarks 12
    - a. Operands 12
    - b. Use of literal as operand 13
    - c. Remarks 13
  - 7. Identification 13
- B. Writing Machine Instructions 13
  - 1. Single Stage Instructions 14
  - 2. Double Stage Instructions 14
- C. Compiler Instruction Format 15

III. DEFINITION OF DATA 16

- A. General 16
- B. Organization of Data 16
- C. Concept of Levels 17
- D. Data Definition 18
- E. Use of the Programming Worksheet for Data Definition 19
  - 1. Page and Line 19
  - 2. Reference 19
  - 3. Operation 19
  - 4. Level 20
  - 5. X 20
  - 6. Length, Type and Remarks 20

TABLE OF CONTENTS (Cont.)

7. Remarks 23
8. Identification 23
F. Addressing of Data 23
1. Relative Nature of 315 Addressing 23
2. Static Addresses 23
3. Dynamic Addresses 24
4. Indexing 24
G. DATA 25
H. INDEX 26
I. REDFN 27
J. CRAM Track Labels 29
K. Variable Length Records 30
1. CRAM 30
2. Magnetic Tape 30

IV. COMPILER CONTROL INSTRUCTIONS 31

A. General 31
B. Description of Control Instructions 31
1. For Definition of Constants 31
a. ALPHA 31
b. DIGIT 32
c. NUMBER 33
d. PAIR 34
e. SLAB 36
f. SGL 36
2. Literal Constants 36
a. ALPHA literal 37
b. DIGIT literal 37
c. Numeric literal 38
d. Reference literal 39
3. To Control the Location Counter 39
a. ORIGIN 40
b. OVRLAY. 41
c. SAVE. 42
d. LITORG 42
4. For Referencing 43
EQUATE 43
5. For Program Addressing 44
a. BASE 44
b. JVAL 45
6. To Modify the Printout 45
a. PAGE 45
b. UNLIST 46
c. LIST 46
7. To Delete a Line During Initial Compilation 46
XXX 46
8. To Delete One or More Lines During a Recompilation 46
OMIT 46
9. For Definition of Data 47
10. To Identify Object Program File Specifications 47
a. FORMAT C 47
b. FORMAT T 47
11. To Identify Compiler Input and Output Specifications 47
NEAT 47
12. To Identify the Last Line Input to the Compiler Run 48
END 48

TABLE OF CONTENTS (Cont.)

13. To Identify the Last Card of a Correction Deck Input to a Recompilation . . . 48
    ENDMOD . . . 48
C. Addressing . . . 48
1. Types of Addresses . . . 48
a. Simple address . . . 48
b. Compound address . . . 49
c. Asterisk address . . . 49
d. Reference literal address . . . 49
e. Positive A or B . . . 50
f. Blank operand . . . 50
2. Descriptive Character of Symbolic Addresses . . . 50
3. To Modify Descriptive Definitions by Using Length and X Entries . . . 51

V. MACRO INSTRUCTIONS . . . 52

A. General . . . 52
B. Writing Macro Instructions . . . 52
C. Scientific Subroutines . . . 53

VI. COMPILER INPUT . . . 54

A. Control Worksheet, Form F-2691 . . . 54
B. CRAM File Tables . . . 59
C. CRAM File Specification Worksheet, Form F-7304 . . . 60
D. FILEC . . . 66
E. Magnetic Tape File Tables . . . 67
F. Magnetic Tape File Specification Worksheet, Form F-7304 . . . 67
G. FILE . . . 73
H. Conventions . . . 74
1. Sequence of Input . . . 74
a. NEAT . . . 74
b. END . . . 74
c. ENDMOD . . . 74
d. FINISHC or FINISH . . . 74
2. Other Requirements . . . 74
a. FORMAT C and FILEC . . . 74
b. FORMAT T and FILE . . . 74
I. Changes . . . 75
J. Deletions . . . 75

VII. COMPILER OUTPUT . . . 76

A. CRAM or Magnetic Tape . . . 76
1. Object Program . . . 76
2. Recompilation Master . . . 76
B. Optional Punched Output . . . 76
1. Punched Paper Tape . . . 76
2. Punched Cards . . . 76
C. Printout . . . 76

VIII. KEYPUNCHING PROCEDURES . . . 80

A. General . . . 80
B. Paper Tape . . . 80
1. General . . . 80

2. Control Worksheet, Form F-2691 81
3. File Specification Worksheet, Form F-7304 81
4. Programming Worksheet, Form F-2689 81
   a. Punching the references 81
   b. Punching the remarks 81
   c. Punching the operands 82
5. Correcting Punching Errors 82
6. End of Paper Tape 82

C. Punched Cards 82
1. General 82
2. Control Worksheet, Form F-2691 83
3. File Specification Worksheet, Form F-7304 83
4. Programming Worksheet, Form F-2689 83
   a. Keypunching the remarks 83
   b. Keypunching the operands 83
5. Correcting Keypunching Errors 84
6. Keypunching Non-IBM Characters 84

IX. INDEX 85

LIST OF ILLUSTRATIONS

Figure No.	Title	Page
1	Run Charts Showing Sequence of Events	7
2	Programming Worksheet	9
3	Control Worksheet, CRAM	56
4	Control Worksheet, Magnetic Tape	57
5	File Specification Worksheet, CRAM	61
6	CRAM File Table Showing Fields Affected by Entries on File Specification Worksheet	62,63
7	File Specification Worksheet, Magnetic Tape	68
8	Magnetic Tape File Table Showing Fields Affected by Entries on File Specification Worksheet	70,71

LIST OF TABLES

Table No.	Title	Page
I	NCR 315 NEAT Compiler Command Formats	15
II	NCR 315 Character Punch Configurations	84

I. INTRODUCTION

The 315 NEAT Compiler is an automatic programming system which will greatly reduce the time and effort required to prepare programs for the NCR 315 Computer. The programmer can use mnemonics and symbolic and relative references to write a source program independent of fixed memory locations; control instructions to direct the compiling process; and macro instructions to call upon an expandable library of macro subroutines. The Compiler will process the source program, assign memory locations to all instructions and data, generate and insert subroutines where indicated by macro instructions, and produce a complete object program in the language of the 315 Computer.

A. EQUIPMENT REQUIREMENT

The 315 NEAT Compiler is designed to be run on an NCR 315 Computer System with the minimal equipment:

A 10,000 slab memory
Five magnetic tape handlers; or one CRAM unit
A paper tape or punched card reader
A high-speed printer

The Compiler will of course operate efficiently on a 315 system with a larger memory (15,000, 20,000, 30,000, or 40,000 slabs), and will compile programs to be run on 315 Computers with any size memory.

B. COMPILER FUNCTIONS

1. Actual Machine Code

A computer program requires: (1) instructions to the computer to process data, and (2) the data which is to be processed.

During the running of the program, both the instructions and the data are stored internally in the computer in numbered locations which have been previously designated to receive this information.

The instructions are composed of several elements, some or all of which are present in every instruction: (1) the operation which the computer will execute, (2) the address of the memory location(s) which contains the data affected by the operation, and (3) in certain situations, the address of the next instruction to be executed.

The computer operations and the memory addresses are expressed by combinations of decimal numbers and other characters which are represented in memory by a series of coded binary digits. It would be impractical for a programmer to attempt to think and write using binary notations. For convenience, the programmer can express his program in decimal numbers which correspond to the operations and to the actual addresses in memory in which the data and instructions are to be stored.

Using this notation, the programmer can write the coded decimal equivalent of the elements of each instruction and of the addresses in memory that are referenced.

2. Automatic Programming

Writing a program using actual machine code and actual memory addresses is time consuming. Also, many errors are possible, both in writing the program on the programming worksheets, and in transcription to punched paper tape or punched cards which serve as input media to the computer. A number of techniques are available to avoid this difficulty, some of which are described as follows.

a. Symbolic coding

For ease in handling, symbols or symbolic names can be assigned to much of the program data. This permits the use of terms or combinations that are an aid to memory, or "mnemonic". The program (called the source program) is punched from the programming worksheets into paper tape or cards and becomes input to a Compiler program which converts the symbols into binary form and produces a program in the language of the computer on which this program will be run. This finished program is called the object program.

1) Operations

Mnemonic symbols can be assigned to machine operations, for example: "1" which represents the operation Load Accumulator can be written "LD", and "57" which represents the operation Jump can be written "JUMP". LD and JUMP are punched into paper tape or cards and are converted by the Compiler into the binary representation of 1 and 57 as required by the computer on which the object program will be run.

2) Addresses

The programmer can assign symbolic names or references (sometimes called tags or labels) to the data, and to the memory addresses of this data. Subroutines or other program points may also be referred to by assigning a name. These names may be mnemonic, may be abbreviations, or may be simply combinations of letters of the alphabet or of letters and numbers. In this way the programmer need not concern himself with where in memory the data is stored; he merely writes the name of the unit as the operand of an instruction. For example: LD AMOUNT. The Compiler will determine the memory location assigned to the AMOUNT field, and will use this address wherever the name AMOUNT appears as the operand of an instruction. If the load instruction entry were assigned the name CALCULATE, then a jump could be taken to this instruction by writing: JUMP CALCULATE.

b. Assignment of Absolute Memory Addresses

The Compiler must assign absolute (actual) memory addresses to all parts of the object program. This includes the assignment of addresses for data input/output areas, all constants used in the program, and all instructions which constitute the object program. To do this, the Compiler utilizes what is normally termed a "location counter". A more detailed explanation of the location counter is presented in Chapter IV.

c. Assembly function

Assume that in the following instructions AMOUNT is a two-slab field located at 03210, and the LD instruction is assigned memory address 04321.

The following entries:

REFERENCE		Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
CALCULATE		LD				AMOUNT
		JUMP				CALCULATE

would be converted into:

XxF C	A
3 1 1	2 1 0

4 5 7	3 2 1

The creation of an object program in absolute form from a series of instructions in symbolic form is called an "assembly" process, and is an inherent feature of the Compiler.

d. Printout

An important function of the Compiler is that it will produce a printout which will include the entire source program and the object program showing the absolute addresses allocated for each instruction, constant, data unit, etc., and the contents of these memory locations.

The printout also contains a listing of errors that have been detected by the Compiler and their point of occurrence. Extensive checks are made throughout the compilation process so that programming errors which can be checked are noted to the programmer.

Another portion of the printout contains a cross-reference listing of all symbolic references used in the program with their absolute addresses, showing where they were defined and where they were used as operands.

The printout is a valuable aid to the programmer. It provides a picture of internal memory and a complete record of his program, and enables him to locate and correct errors in his program and to make whatever additions, deletions, or substitutions may be necessary.

e. Remarks

Another useful feature of the Compiler is the ability to carry "Remarks". These are notes written on the programming worksheet by the programmer and punched with the instructions, and appear on the printout. The remarks perform no function for the Compiler but are useful for documentation and checking purposes.

f. Control instructions

In writing a program to be compiled, the programmer uses symbols in lieu of actual machine operations and addresses. This produces a program in an artificial or "pseudo" language. These pseudo codes are in a form that is similar but not identical to the machine code format for the instructions which they represent.

In addition to representing machine instructions, pseudo language can be used to supply special information which will be required by the Compiler. These directives--called "control instructions"--while affecting the compiling process, will not introduce any instructions into the object program. Control instructions

are used to describe data and constants, and to direct the Compiler in the allocation of memory space and assignment of addresses.

For example:

REFERENCE	18	Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
AMOUNT		NUMB	ER	2		1223

The control instruction NUMBER directs the Compiler to place the numerals 1223 right justified in two slabs of memory to be referred to as AMOUNT. Assuming that at the time the instruction is converted into absolute code, the next available location in memory is 03210, the Compiler will allocate memory location 03210 and 03211 and will remember that the length of AMOUNT is two slabs and contains:

03210	03211
0 0 1	2 2 3

When the reference AMOUNT is entered as the operand of an instruction, the Compiler will use the address and length of AMOUNT in creating the absolute code.

For example:

REFERENCE	18	Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
		LD				AMOUNT

becomes:

X xF C	A
3 1 1	2 1 0

Thus, the programmer can write AMOUNT to indicate the address of the memory location containing the value 1223. In this way, he need not be concerned with the length of this particular constant, or where in memory the constant will be stored.

The use of a Compiler program provides many benefits. The source program becomes easier to write. Mnemonics are easier to learn and to use than are machine codes. Specific memory addresses can be ignored completely. There are fewer possibilities of error in the writing of a program and in its conversion to punched paper tape or punched cards. Less time is required between the definition of a problem and its production running. Several programmers can work together writing separate parts of a program without loss of time or efficiency.

3. Macro Instructions

One of the most powerful features of the Compiler is the ability to call upon a library of macro subroutines. These subroutines exist in a generalized form as part of a macro generator which is stored in a library (either on CRAM or on magnetic tape).

The programmer simply writes a line of coding (referred to as a macro instruction) on the programming worksheet, at the point where the subroutine is to be executed in the object program. The macro instruction consists of the name of the subroutine and any parameters (entered as operands) that are required so that the subroutine can execute its function for this particular program. The Compiler will obtain the desired macro generator, identified by the macro name, from the library, and will transfer control to it.

The generator contains within itself a subroutine in skeleton form. The generator selects the appropriate sections of the subroutine and modifies individual instructions to suit the parameters. These instructions are generated in symbolic form and the Compiler will assign the appropriate memory locations to this completed subroutine and convert the instructions to absolute form.

4. Scientific Subroutines

For scientific applications, a Compiler is available having a complete package of subroutines necessary to perform higher order mathematical formulae. Examples of these subroutines include calculation of square root, complex addition and normalize, floating decimal operations, integration, algebraic operations, and many others.

C. FILE TABLES

An advanced technique employed by the NEAT System is the use of file tables. These tables are created by the Compiler and will be held in memory during the running of the object program. They will contain the parameters pertinent to the various CRAM or magnetic tape files, and will also be used to store data generated during the running of the object program. In writing the macro instructions in the source program, in most cases it is only necessary to enter the reference to the file table for a particular file. The macro generator and the Compiler will supply the appropriate addresses of the fields in the file table and insert them in the machine instructions in the generated subroutine.

D. EXECUTIVE ROUTINES

A complete package of executive routines is provided to automatically handle the many repetitive and routine situations encountered during the running of a program. This includes file label checking, end-of-file procedures, read-write error procedures, rescue point procedures, overlay control, and other important routines. These routines make considerable use of the file tables.

CRMX-II is the executive system provided by NCR for programs using CRAM

STEP (Standard Tape Executive Program) is the executive system provided by NCR for programs using magnetic tape.

E. LIBRARIAN

Object programs produced by the Compiler may be used to add to, delete from, or change the program library. These functions are performed by a CRAM Librarian for programs on CRAM, or by a Magnetic Tape Librarian for programs on tape.

F. PROCEDURE

The programmer must provide the Compiler with all the information required to produce the object program. This information will be in the form of machine instructions, control instructions, macro instructions, constants, remarks, file table information, etc., which constitute the source program. The Compiler will use this source program to produce the object program.

Following is a brief statement of some of the steps involved in using the 315 NEAT Compiler.

1. The programmer writes the source program in the form of entries in a specified format on printed forms designed for convenience in writing and keypunching.

2. Entries on these forms are keypunched into paper tape or cards.

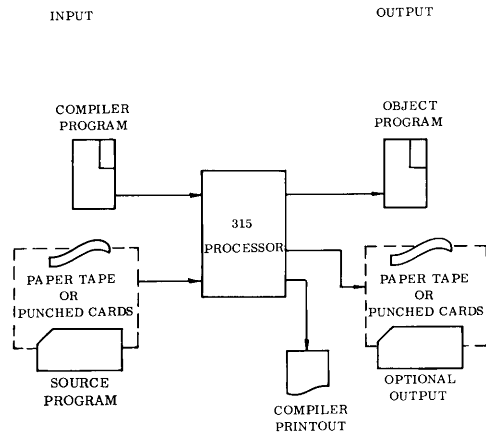
3. The punched paper tape or cards become input to a computer Compiler run. The main output of this run is the object program on CRAM or on magnetic tape. The object program may also be output on punched cards or paper tape.

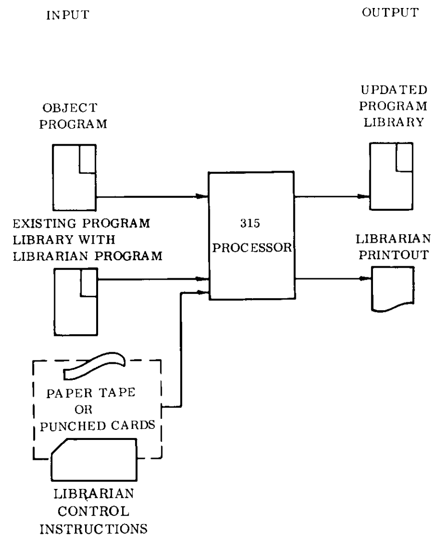
4. The object program on CRAM or magnetic tape becomes input to a computer Librarian run. This run incorporates the program as a part of a new Program Library, making any changes to the program as directed by the programmer.

5. The object program on the Program Library becomes input to a computer run for processing data. The output of this run may be one of the following, depending upon the application and equipment involved: CRAM, magnetic or paper tape, punched cards, printout, or inquiry.

G. SEQUENCE OF EVENTS

Figure 1 is a set of run charts showing the sequence of events normally involved in compiling an object program which will be used to process data. In the figure, a CRAM system is shown; however, a magnetic tape system may be used as well. The figure illustrates the compatibility of the NCR software programs, showing how the output of one program becomes the input of the next program. It should also be noted that, for simplicity of illustration, only the minimum number of CRAM units necessary to convey the functions of input and output are shown. This is not necessarily the actual number of CRAM units required, as this will vary, depending upon the individual user's requirements.





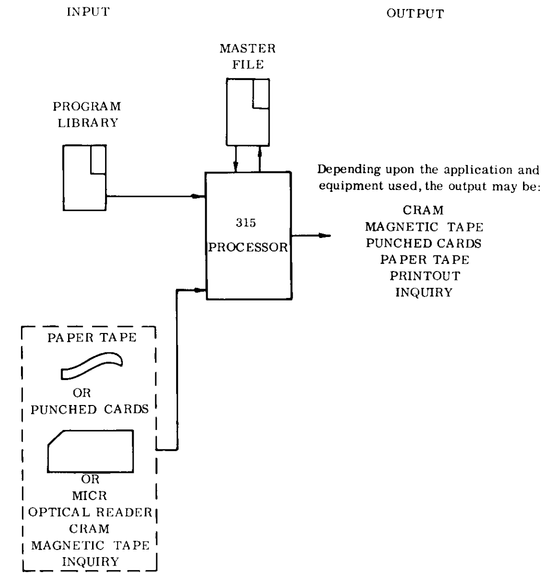


Figure 1. Run Charts Showing Sequence of Events

II. EXPLANATION OF THE PROGRAMMING WORKSHEET

A. PROGRAMMING WORKSHEET

The Programming Worksheet, Form F-2689, is provided for convenience in writing and keypunching the source programs. Each line on the sheet has the maximum capacity of 80 characters, corresponding to the 80 columns of a punch card or 80 characters punched into paper tape. In general, an entry on one line of the programming worksheet will represent one instruction in the source program.

The following is a brief description of the entries in the various columns of the programming worksheet.

1. Page and Line (columns 1-6)

The Page and Line columns are used to indicate the sequence in which lines of coding are to be compiled. They may contain any of the 64 characters from the 315 Code Chart; however, if a sort is to be performed externally, characters must be used which will be recognized by the sort equipment being used.

A unique page number may be assigned to each programming worksheet, or one page number may include several sheets. The line numbers are assigned in ascending alphanumeric sequence within the page number. Each line of coding is assigned a line number.

Page and Line entries should consist of three characters each. Spaces (or punched card blanks) should not be used unless it is intended that they have their literal value. If no value is desired, leading zeros should be entered. For example:

LINE		
4	5	6
	9	0
1	0	0

should be written

LINE		
4	5	6
0	9	0
1	0	0

Otherwise, an out-of-sequence condition will result since ⌀ 9 0 has a greater alphanumeric (binary) value than 1 0 0. (The symbol ⌀ represents the character "space".)

In assigning page and line numbers, allow for the possible insertion of additional lines of coding. This can be done simply by leaving a gap between numbers on subsequent lines. For example, the assignment:

LINE		
4	5	6
1	1	0
1	2	0
1	3	0
1	4	0

would permit nine additional lines (counting numerically) to be inserted between any two

315 NEAT* COMPILER

PROGRAMMING WORKSHEET

Program ______________________

Prepared by ______________________

Date ______________________ Page ____ of ____

PAGE	LINE		REFERENCE		Op ¦ V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE	REMARKS / REMARKS	IDENTIFICATION
1 2 3	4 5 6	7	8 9 10 11 12 13 14 15 16 17	18	19 20 21 22 ¦ 23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48	49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74	75 76 77 78 79 80
1 2 3	4 5 6	7	8 9 10 11 12 13 14 15 16 17	18	19 20 21 22 ¦ 23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48	49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74	

F-2689 *TRADEMARK *SERVICE MARK PRINTED IN U.S.A.

Figure 2. Programming Worksheet

existing lines without the need to renumber or to repunch any lines. If letters and other characters, as well as numbers, are being used for Line entries, this would provide 63 additional lines.

The Page-Line column may be left blank; however, this is not recommended.

2. Reference (columns 8-17)

In writing the source program extensive use can be made of symbolic names and references. These names (sometimes called tags or labels) are assigned to identify the data which is to be processed, and the constants and working storage required by the program, as well as instructions and other lines of coding to which the program will refer. A name can then be used as a "reference" to the data, or memory location, etc., which it identifies. Thus the reference is entered as the operand of an instruction rather than the actual memory address.

A reference may be from one to ten characters long and may be made up of any of the letters of the alphabet (A to Z) and any of the decimal numerals (0 to 9) but must contain at least one letter. A name may not contain a space.

> There is one exception to this rule: that is in the assignment of program names. See Program Names, Chapter V.

Names may be mnemonic or abbreviations, or simply arbitrary combinations of letters of the alphabet or letters and numbers. For example, a Transaction File could be named:

TRANSFILE   or   TRFILE   or   F3

The Compiler will equate the reference with the address of the coding or the memory area resulting from this line. If the line contains a control instruction which sets up a constant or defines a data unit, the Compiler will also associate the length and the characteristics of the coding or memory area resulting from this line. Thus, a reference is said to be "defined".

A reference name used as the operand of an instruction must be defined once and only once (must appear once in the Reference column of some line).

Since references must be unique, no two lines of coding may be assigned the same name.

At times, it may be convenient to assign a name merely for identification even though it may never be used for reference elsewhere in the program.

Reference entries must be left justified. Maximum field size is 10 alphanumeric characters.

Note: An asterisk (*) entered in column 8 (the first column of the Reference field) signifies that the entry on this line is to be treated as a remark. See Remarks.

3. Operation (Op :V) (columns 19-24)

The Operation column (designated Op :V) is used to indicate the purpose of the instruction represented by that line of coding. There are three classes of instructions available for use with the Compiler. These are: (1) 315 machine instructions, (2) control instructions, and (3) macro instructions.

a. 315 machine instructions

These instructions specify particular computer operations, e.g., Load Accumulator, Add to Accumulator, Store Accumulator, Add to Memory, etc. The operation portion of a machine instruction is written using the mnemonic form, e.g., LD, ADD, ST, ADD M, etc.

The mnemonic operation code is entered left justified in the Op portion of the column, and the variation code, if applicable, is entered left justified in the V portion. Machine instructions are described in the 315 Programming Handbook, MD 315-02.

b. Control instructions

These instructions, e.g., DATA, ALPHA, SAVE, END, etc., facilitate the definition of data and constants, assignment of memory locations, and otherwise control the compiling process.

Control instruction mnemonics are entered left justified in the Operation column. Control instructions are described later in this manual.

c. Macro instructions

These instructions, e.g., MLDR, DYDUMP, NEXTIN (for CRAM), or NEXTI (for magnetic tape), etc., are each replaced by the Compiler with a series of instructions which will be generated using the library of macro subroutines.

Macro instruction mnemonics are entered left justified in the Operation column. Macro instructions are described in the 315 Macro Instructions Manual, MD 315-44.

Caution: The Operation entries must be spelled and written correctly. The Compiler will compare the entry with lists of instruction names. If the entry does not correspond exactly to any in the lists, it will be treated as an error.

4. Length (columns 25 and 26)

Length entries represent the length of the operands in slabs and may be either left justified or right justified. Length entries must be numeric. Symbolic entries are not permitted in this field.

Some single stage machine instructions (LD, ST, ADD, etc.) normally require a Length entry; however, with the Compiler, length need be entered only where the operand is an absolute address.

The Length column can usually be left blank since most instructions in the Compiler do not require a length entry.

Length is not required in a single stage machine instruction where the operand is a symbolic reference. The Compiler will use the length associated with the descriptive definition of the symbolic operand; that is, the length assigned at the time the data unit or the constant was defined.

If a length is entered with a symbolic operand, then for that instruction only, the Compiler will use this new length in place of that associated with the operand. This permits the temporary modification of the length of a machine instruction operand, but only for the instruction containing the modifying Length entry. If Length is left blank the next time the symbolic operand is used, the Compiler will use the length as originally defined.

Caution: A blank length field with a compound operand field (e.g. AMOUNT + TAX) will be assigned the length associated with the first term (sic. AMOUNT).

Length is never required with any double stage machine instruction (MOVE:B, MOVE:E, SAUG:R, etc.).

Length is not required with most control instructions--except ALPHA, DIGIT, and NUMBER.

Length is usually not required with macro instructions.

5. X (columns 27 and 28)

   X entries usually represent index registers associated with the operands and may be either left justified or right justified.

   X is used mainly with Data Definitions, with machine instructions having absolute operands, and occasionally with some of the control instructions or some of the macro instructions.

   X, like Length, can usually be left blank. X may also be symbolic.

   X is not required in machine instructions with symbolic operands, and if entered, the Compiler will use this index register for this one instruction, instead of the index register originally associated with the operand.

6. Operands and Remarks (columns 29-74)

   This is a combined column containing Operand and Remark entries as indicated below.

   a. Operands

      The entries in the Operands portion of the column are determined by the characteristics of the instruction.

      Operand entries are left justified. When an instruction requires several operands, they are written as a continuous statement, separated only by commas. Where a particular operand of an instruction is not required, it is not entered; however a comma is entered in its place, if it is followed by another operand. Thus trailing commas can be omitted.

      Operands may not contain spaces nor may there be spaces between operands, and an Operands entry may not be extended to the next line. (Macro instructions are the exception to this rule. See Writing Macro Instructions, Chapter V.)

      Two consecutive spaces occurring in this column denote to the Compiler that this is the end of the Operands entry and that Remarks follow.

      References may be used as operands of instructions. Normally the reference could either have been defined previously in the source program, or in some later portion. However, there are four control instructions (ORIGIN, OVRLAY, SAVE, and EQUATE) which require that where a reference is used as an operand, it must have been defined earlier in the source program.

      If an asterisk (*) is entered as an address, it represents the address of the first slab of coding generated by that line.

      Caution: Absolute addresses are permitted as operands; however, great care should be taken since their correct use requires a thorough knowledge of the Compiler program and of the CRMX-II or STEP routines. Symbolic operands are more flexible and much more convenient to use.

In writing an absolute address, leading zeros may be omitted. For example, the entry 123 would be treated by the Compiler as 00123.

b. Use of Literal as Operand

A literal is a unit of data whose value is identical to those characters composing the unit. A literal may be entered as an operand by writing in the Operand column the number sign "#" and a letter designating the type of literal, followed by the actual value enclosed in parentheses. One-slab literals will be compiled as the operand of the instruction for those instructions that permit it. Literals larger than one slab will be placed in a storage area (called the Program Safe Area) and the address of this literal will be stored as the operand of the instruction. Writing of literals is explained later in greater detail. (See Literals, Chapter IV.)

c. Remarks

The information entered in the Remarks portion of the column is punched and will be included by the Compiler in the source program listing in the printout, but is not treated as part of the object program. Remark entries may contain any character in the 315 Code Chart, including spaces.

Remark entries are usually comments which may be helpful in documenting and checking the program. For example, remarks may be used to explain the functions of the program or to state the reason for a particular instruction when its purpose is not obvious.

As was stated above, two spaces (blanks) indicate that the following are remarks. Remarks may begin anywhere in the column provided there are at least two spaces between the last operand and the remarks. However, a listing is much easier to read if the lines have a common left margin, and for this purpose, column 49 of the programming worksheet is marked with a vertical broken line. For uniformity, and for convenience in keypunching and reading, remarks should begin at column 49. But this is merely a convention, and if the last operand extends past column 47, leave two spaces, then start the remarks, if any.

A long Remarks entry that would extend past column 74 can be continued on the following lines. Simply assign the next line number, write an asterisk (*) in column 8, and continue the remarks.

Note: An asterisk (*) entered in column 8 (the first column of the Reference column) signifies that the entire line is to be considered a remark. In this case, the remark may begin anywhere on the line from column 9 on.

7. Identification (columns 75-80)

The Identification column is not used with punched paper tape.

When punched cards are used, an entry in this column permits a six character identification to be punched into each card so that it will be possible to visually identify the program to which the card belongs. However, identification is for external use only. The Compiler does not check these columns.

Any character in the 315 Code Chart may be used.

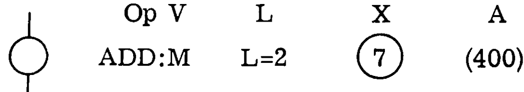
B. WRITING MACHINE INSTRUCTIONS

Machine instructions are written on one line of the programming worksheet. Entries are made according to the following pattern:

Column	Entry
Op	Operation code
V	Variation code
Length	Length
X	Index Register
Operands	A operand (then B and Y operands if double stage instruction)

1. Single Stage Instructions

For example: The instruction to add the contents of the accumulator to a two-slab memory word beginning at address 400 relative to index register 7, could be shown on the flow chart as:



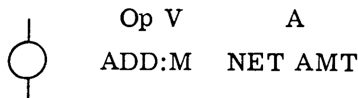
The corresponding entry on the programming sheet would be:

Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE	REMARKS / REMARKS
19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48	49 50 51 52 53 54 55 56 57 58 59 60 61
A D D	M	2	7	4 0 0	

This would be converted by the Compiler into the absolute format of a single stage instruction and stored in two slabs as:

X xF C	A
7 1 9	4 0 0

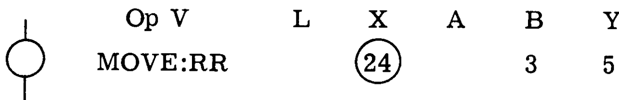
Actual values may be used to designate length, index register, and memory address for each instruction, as shown above. However, it is more common and more advantageous to use the various symbolic and relative addressing tools provided by the Compiler. Thus, this instruction could be written simply:



Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45
A D D	M			N E T A M T

2. Double Stage Instructions

For double stage instructions, the A, B, and Y operands are written as a continuous statement, separated only by commas. No spaces are permitted between operands of machine instructions. Some instructions do not require entries for all of the operands. The absence of an operand entry in a double stage instruction is indicated by a comma. For example:



Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
M O V E	R R		2 4	, 3 , 5

X xF C	A
Y yQ G	B
8 @ 0	0 0 0
5 0 4	0 0 3

C. COMPILER INSTRUCTION FORMAT

The following table shows the format of the instructions used in the Compiler program.

Op	V	L	X	Operand
*ADD		L	X	A
*ADD	M	L	X	A
AUG	J		X	A,N,Y
AUG	R		X	A,N,Y
*BADD		L	X	A
BACK				,I,Y
*CLRF	LH		X	A
*CLRF	RH		X	A
*CLRU	C		X	A
*CLRU	P		X	A
*CLRU	Q		X	A
*CLRU	S		X	A
*CNT			X	A,G,Y
*COMP		L	X	A
*DIV		L	X	A
DLR				
EDIT		L	X	A
HALT	A		X	A
HALT	D		X	A
JUMP			X	J
JUMP	I		X	A
JUMP	IP		X	A
*LD		L	X	A
LD	J		X	A,N,Y
LD	R		X	A,N,Y
LDAD			X	A,L,Y
LDAD	XB		X	A,L,Y
LDAD	XL		X	A,L,Y
LDAD	XR		X	A,L,Y
MLRA				G
MOVE	B		X	A,B,Y
MOVE	E		X	A,B,Y
MOVE	JJ		X	,N,Y
MOVE	JR		X	,N,Y
MOVE	RJ		X	,N,Y
MOVE	RR		X	,N,Y
*MULT		L	X	A

Op	V	L	X	Operand
PAST	XB		X	A,L
PAST	XL		X	A,L
PAST	XR		X	A,L
PKT			X	A,,Y
PNCH			X	A,S,Y
PPT	C		X	A,N,Y
PPT	S		X	A,N,Y
PRNT			X	A,MF,Y
RCC			X	A,I,Y
RCK			X	A,,Y
RCOL			X	A,N,Y
RCOL	F		X	A,N,Y
RCOL	T		X	A,N,Y
RCOL	TF		X	A,N,Y
RMT			X	A,I,Y
RPT	C		X	A,N,Y
RPT	CX		X	A,N,Y
RPT	S		X	A,N,Y
SAUG	J		X	A,N,Y
SAUG	R		X	A,N,Y
SCNA	v		X	A,L,Y
SCND	v		X	A,L,Y
SELC	DN		X	A,J,Y
SELC	DP		X	A,J,Y
*SELC	R		X	A,J,Y
*SELC	T		X	A,J,Y
*SELP			X	A,J,Y
*SELQ			X	A,J,Y
*SELS			X	A,J,Y
SETF	+			
SETF	-			
SETF	O			
SETF	D			
SETF	T			
*SETF	LH		X	A
*SETF	RH		X	A
*SETU	C		X	A
*SETU	P		X	A
*SETU	Q		X	A
*SETU	S		X	A

Op	V	L	X	Operand
*SHFT	AL		X	A
*SHFT	AR		X	A
*SHFT	DL		X	A
*SHFT	DR		X	A
*SHFT	RR		X	A
*SHFT	RC		X	A
*SHFT	LC		X	A
SKIP				G
SLD	J		X	A,N,Y
SLD	R		X	A,N,Y
SPRD	B			#,B,Y
SPRD	E			#,B,Y
*ST		L	X	A
ST	J		X	A,N,Y
ST	R		X	A,N,Y
STDA			X	A,L
STRT	S			
STOP	S			
*SUB		L	X	A
SUPP		L	X	A
TEST	E		X	J
TEST	G		X	J
TEST	L		X	J
TEST	-		X	J
TEST	O		X	J
TEST	D		X	J
TEST	T		X	J
*TEST	LH		X	A,J
*TEST	RH		X	A,J
*TEST	SW		X	A,J,Y
TYPE	A		X	A,N
TYPE	AP		X	A,N
TYPE	D		X	A,N
WIND				,I,Y
WIND	L			,I,Y
WCC			X	A,I,Y
WMT			X	A,I,Y

TABLE I. NCR 315 NEAT COMPILER COMMAND FORMATS

* means A may be a literal

means A is required to be a literal

III. DEFINITION OF DATA

A. GENERAL

In generating the object program, the Compiler must allocate memory area for data to be referenced and must set up a precise addressing pattern for the fields contained in the data. This must be done for files that are read into memory from an external medium (CRAM, magnetic tape, punched paper tape, punched cards), for files that are written out to an external medium from memory, and for files that are manipulated while in memory. This must also be done for other data such as tables that are referenced in memory. To do this, the Compiler requires a description of the size and characteristics of this data as well as the memory location. This descriptive process is called Data Definition.

B. ORGANIZATION OF DATA

An important function of a data processing system is the creation and manipulation of files. A file is composed of a set of related records (sometimes called logical records). A record contains pertinent information about a common subject. Records are transferred to and from CRAM or magnetic tape in the form of blocks (sometimes called physical records). A block usually contains several records; however, it may at times contain only one record or a part of a record.

The most basic subdivision of a record is called a field. A field is composed of successive characters which specify a particular unit of information. Each field is a separate entity; however, this same field may also be a part of another field, and it may similarly include other fields within its own definition.

For example, a three-slab field containing INVENTORY DATE would be treated as a single unit of information.

INVENTORY DATE

XX	XX	XX

If the fields within INVENTORY DATE are to be referenced individually, it would be treated as having three fields of one slab each: MONTH, DAY, and YEAR. Thus, MONTH would be assigned the same memory address as INVENTORY DATE.

INVENTORY DATE

Month	Day	Year
XX	XX	XX
007	008	009

If only MONTH and DAY are to be referenced individually, YEAR need not be specifically defined; however, its presence should be noted since it is included in INVENTORY DATE.

INVENTORY DATE

Month	Day	
XX	XX	XX
007	008	009

For the purpose of Data Definition, a field will be referred to as a "data unit". The term data unit will also be applied to a record and to a block. Each data unit that will be referenced must be assigned a unique name.

List the named data units in the order in which they occur; that is, a unit, any sub-units, then the next unit, indenting each time a division or subdivision occurs, and entering the length

of each unit. Where a unit is not divided, return to a similar previous indentation. This indented listing will help show the relationship of any data unit to any other data unit.

Example: A Parts Procurement File is composed of blocks of records containing an inventory and status of manufactured parts. Each block contains 70 records; each record contains 20 slabs.

PARTS RECORD																			
PART NUMBER				DESIGN DATE			INVENTORY DATE			STATUS		(Quantity Manufactured in Each Quarter)							
SERIAL NUMBER			CODE				MON	DAY		IN PROCESS		QMEQ		→					
X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X
000	001	002	003	004	005	006	007	008	009	010	011	012	013	014	015	016	017	018	019

Data Unit	Length
PARTS BLOCK	1400 slabs
PARTS RECORD	20 slabs
PART NUMBER	4 slabs
SERIAL NUMBER	3 slabs
CODE	1 slab
DESIGN DATE	3 slabs
INVENTORY DATE	3 slabs
MON	1 slab
DAY	1 slab
(not named)	1 slab
STATUS	10 slabs
IN PROCESS	2 slabs
QMEQ	2 slabs
QMEQ	2 slabs
QMEQ	2 slabs
QMEQ	2 slabs

C. CONCEPT OF LEVELS

If we examine the record layout, we can see that the arrangement of units suggests a data hierarchy. The block occupies the highest level, the record is next, then a number of units. Some of these units are broken down still further into sub-units.

This concept of levels, which is inherent in the structure of a record, is also represented in the indented listing. In order to express these logical indentations to the Compiler program, we can use a system of level numbers. The most inclusive unit would be assigned the level number 1, the next level would be level 2, then 3, 4, etc., in the manner in which we indented to show a lower level division. Where a unit is not divided, assign the same level number that was assigned to a previously named unit which occupies the same relative position in the hierarchy, for example PART NUMBER and DESIGN DATE.

2 PARTS RECORD																			
3 PART NUMBER				3 DESIGN DATE			3 INVENTORY DATE			3 STATUS									
4 SERIAL NUMBER			4 CODE				4 MON	4 DAY	4	4 IN PROCESS		4 QMEQ		→					
X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X
000	001	002	003	004	005	006	007	008	009	010	011	012	013	014	015	016	017	018	019

Data Unit	Level	Length
PARTS BLOCK	1	1400 slabs
PARTS RECORD	2	20 slabs
PART NUMBER	3	4 slabs
SERIAL NUMBER	4	3 slabs
CODE	4	1 slab
DESIGN DATE	3	3 slabs
INVENTORY DATE	3	3 slabs
MON	4	1 slab
DAY	4	1 slab
(not named)	4	1 slab
STATUS	3	10 slabs
IN PROCESS	4	2 slabs
QMEQ	4	2 slabs
QMEQ	4	2 slabs
QMEQ	4	2 slabs
QMEQ	4	2 slabs

D. DATA DEFINITION

A complete definition of a data unit includes the following information:

Name of the data unit (reference)
Operation code (DATA and INDEX, and where applicable REDFN)
Level number
Index register number (in some cases assigned by the Compiler)
Length of the data unit

If the Manufactured Parts Record is to be defined, first complete a Record Layout Worksheet (Form F-2074) using the actual names assigned (to be used as references) to the data units.

NUMBER OF SLABS PER RECORD: 20 NUMBER OF RECORDS PER BLOCK: 70

REFERENCE	I-REG.	BASE
01 PARTSBLOCK	10	
	11	

FILE NAME Parts Procurement File RECORD NAME Manufactured Parts Record

<table>
<tr><td colspan="20">02 PARTSREC</td></tr>
<tr><td colspan="4">03 PARTNUM</td><td colspan="3">03 DESIGNDATE</td><td colspan="3">03 INVENDATE</td><td colspan="10">03 STATUS</td></tr>
<tr><td colspan="3">04 SERIALNUM</td><td>04 CODE</td><td colspan="3"></td><td>04 MON</td><td>04 DAY</td><td></td><td colspan="2">04 INPROCESS</td><td colspan="2">04 QMEQ</td><td colspan="2"></td><td colspan="2"></td><td colspan="2"></td></tr>
<tr><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td><td>XX</td></tr>
<tr><td>000</td><td>001</td><td>002</td><td>003</td><td>004</td><td>005</td><td>006</td><td>007</td><td>008</td><td>009</td><td>010</td><td>011</td><td>012</td><td>013</td><td>014</td><td>015</td><td>016</td><td>017</td><td>018</td><td>019</td></tr>
</table>

Note that there is no entry in the Base column since it is not known where in memory the Compiler will store the block. The actual address of PARTSBLOCK can be obtained by referring to the listing of references in the printout.

Now transfer the information from the record layout worksheet to the programming worksheet. The following illustration shows the actual Data Definition of PARTSBLOCK (a block containing a series of Manufactured Parts Records). The areas printed in grey (DATA, INDEX, etc.) are also part of the Data Definition and will be explained later in this chapter.

REFERENCE (8-17)	Op (19-22)	V (23-24)	LENGTH / LEVEL (25-26)	X (27-28)	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE (29-48)	REMARKS (49-53)
PARTSBLOCK	DATA		0 1		1 4 0 0 S	
	INDEX				1 0 , 1 1	
PARTSREC	DATA		0 2	1 0	2 0 S	
PARTNUM			0 3		4 S	
SERIALNUM			0 4		3 S	
CODE			0 4		1 S	
DESIGNDATE			0 3		3 S	
INVENDATE			0 3		3 S	
MON			0 4		1 S	
DAY			0 4		1 S	
STATUS			0 3		1 0 S	
INPROCESS			0 4		2 S	
QMEQ			0 4	1 1	2 S	

E. USE OF THE PROGRAMMING WORKSHEET FOR DATA DEFINITION

Data Definitions may occur anywhere in the source program. Beginning with the next available slab, the Compiler will allocate memory for the size shown for the entire block, and will assign addresses to data units within the block according to their description (definition) as shown on the programming worksheet.

The following is a brief description of the entries in the various columns of the programming worksheet when it is used for Data Definition.

1. Page and Line (columns 1-6)

   Entries in the Page and Line columns are the same as described in Chapter II.

2. Reference (columns 8-17)

   An entry in the Reference column becomes the name (reference) of the data described on that line. If the data described will not be referenced directly elsewhere in the program, a name need not be assigned.

   Reference entries must be left justified. The rules for assigning references are stated in Chapter II.

3. Operation (columns 19-24)

   There are three control instructions which are used for Data Definitions:

   DATA — indicates to the Compiler that this and the following lines are Data Definitions.

   INDEX — indicates to the Compiler that the index registers listed will contain the base address of the data unit described on the next line.

   REDFN — indicates to the Compiler that the previous data unit with the level named will be redefined and the corresponding memory area will be reallocated to the data units described on the following lines.

Only these three control instructions may be used within a set of Data Definitions. Any other instruction will break the sequence of the definition and result in incorrect level and memory space assignments.

Operation entries must be left justified. The use of these control instructions is described later in this chapter.

4. Level (columns 25 and 26)

   Entries in the Level column represent the level number assigned to the data unit described on this line.

   The most inclusive data unit is assigned level number 1. Lesser units are assigned level 2, 3, 4, etc. A level includes all the data units described under it which have level numbers larger than its own.

   The numbers 1, 2, 3, 4 in the example could have been 1, 2, 4, 6 or 1, 3, 5, 7 or any combination, provided not more than one level number is skipped between successive lower level entries. However, for simplicity, it is recommended that the pattern 1, 2, 3, 4, etc. be used.

   The Compiler will reserve memory according to the length shown for higher level data units. This eliminates the need to describe every slab of every data unit in the block. For example, although the record PARTSREC occurs 70 times in memory (PARTSBLOCK contains 70 records), it need be described only once. The Compiler will reserve 1400 slabs for the entire block according to the level 1 entry. Note also that slab 9 is not shown as a separate level 4 entry. The Compiler will still be able to assign the correct address for INPROCESS since the length shown for INVENDATE is three slabs.

   Level entries may be either left justified or right justified. Level numbers 1 through 9 may be written as a single numeral (1, 2, 3, etc.), or as two numerals (01, 02, 03, etc.).

5. X (columns 27 and 28)

   Entries in the X column designate the index registers to be used by the Compiler when constructing references to this data unit. This includes the data unit described on this line and all lower level units that are included in its definition until a different index register number occurs in the X column.

   Index register numbers need not be repeated for lower level data units. If the X column is left blank, the Compiler will use the last named index register. If no register has been designated, the Compiler will construct addresses using the index register that contains a value within 1000 of the address.

   If a data unit has a fixed location in memory, there is no need to designate an index register for it. However, if the data unit is repeated (e.g. elements of an array, such as records in a block or values in a table), an index register must be designated in order to step through the data units, incrementing the index register by the size of the unit.

   Entries in the X column may be either left justified or right justified.

6. Length, Type and Remarks (columns 29-74)

   This is a combined column containing Length, Type, and Remark entries as indicated below.

   a. The first entry in this column indicates the length (size) of the data unit. The Compiler will use the Length entries to allocate memory and to calculate addresses for the data units. Length may be stated in one of three ways:

1) An integer followed by an S.
   This states the length of the data unit in slabs. For example, the entry 2S indicates the length is 2 slabs.

2) A simple integer.
   This states the length in characters (Alpha form--6 bits). For example, the entry 4 indicates the length is 4 Alpha characters.

3) A mixed number containing a decimal point.
   This states the length in Digits (4 bits) and indicates a number of fractional positions. For example, the entry 4.2 indicates a 6 Digit number having 2 fractional positions. The entry 6.0 indicates a 6 Digit number having no fractional positions. The entry 0.6 indicates a 6 Digit number having 6 fractional positions. The decimal point is punched but does not occupy any space in memory.

The first example (2S) will reserve 2 slabs of memory. The second example (4 representing 4 Alphas) and the third example (4.2 or 6.0 or 0.6 representing Digits) can also reserve 2 slabs, and will supply additional information regarding the data units. This additional information is useful for documentation purposes, and will enable the programmer to specify to the Compiler how the data is to be packed.

The Compiler will reserve an integral number of slabs in memory according to the length stated for level 1 entries. Lower level data units will be stored left justified as permitted within the area reserved for its next higher level.

In the following example, note that the Compiler will allocate two slabs for each data unit even though each one has been described differently (2 slabs, 4 Alphas, 6 Digits).

REFERENCE (8–17)	18	Op (19–22)	V (23–24)	LENGTH / LEVEL (25–26)	X (27–28)	INSTRUCTIONS: / DATA DEFINITIONS: (29–35)
ENTRY				02		6S
NUMERAL				03		2S
CLASS				03		4
RATING				03		4.2

02 ENTRY					
03 NUMERAL		03 CLASS		03 RATING	
		XX	XX	XXX	XXX
000	001	002	003	004	005

The instructions:

Op (19–22)	V (23–24)	LENGTH / LEVEL (25–26)	X (27–28)	INSTRUCTIONS: / DATA DEFINITIONS: (29–35)	
LD				NUMERAL	references a 2-slab unit beginning at slab 000
LD				CLASS	references a 2-slab unit beginning at slab 002
LD				RATING	references a 2-slab unit beginning at slab 004

However, if NUMERAL had been described as 3 Alphas, the following assignment would result:

REFERENCE 8–17	Op 19–22	V 23–24	LENGTH / LEVEL 25–26	X 27–28	INSTRUCTIONS: / DATA DEFINITIONS: 29–35
ENTRY			02		6 S
NUMERAL			03		3
CLASS			03		4
RATING			03		4 . 2

ENTRY					
NUMERAL					
X X	X				
000	001	002	003	004	005

	CLASS				
	X	X X	X		
000	001	002	003	004	005

			RATING		
			X	X X X	X X
000	001	002	003	004	005

The instructions:

Op 19–22	V 23–24	LENGTH / LEVEL 25–26	X 27–28	INSTRUCTIONS: / DATA DEFINITIONS: 29–35	
LD				NUMERAL	references a 2-slab unit beginning at slab 000 which also contains part of CLASS
LD				CLASS	references a 2-slab unit beginning at slab 001 which also contains part of NUMERAL and part of RATING
LD				RATING	references a 3-slab unit beginning at slab 003 which also contains part of CLASS plus an excess Digit

Data may be packed in this manner; however, in this case it is necessary to manipulate the data units by using appropriate machine instructions, LDAD, STDA, PAST, etc., in order to process the data correctly.

b. The second entry in this column indicates the type of the data unit. Any one- or two-character code which is understood by the macro generator subroutines may be entered here. Some possible types are:

A = Alpha (6 bits)
D = Digit (4 bits)
N = numeric (0 through 9)
+N = numeric, guaranteed positive
FP = floating point

The Type entry is optional. If entered, it will be carried by the Compiler in the descriptive definition of the data unit. The Type option is provided for use by future macros.

A Type entry is separated from the Length entry by a comma. For example: 4.2,+N

Length and Type entries are left justified. No spaces are permitted. As with machine instructions, two spaces occurring in this column denote to the Compiler that this is the end of the entry and that Remarks follow.

7. Remarks

The rules for Remark entries are the same as stated in Chapter II.

8. Identification (columns 75-80)

The rules for Identification entries are the same as stated in Chapter II.

F. ADDRESSING OF DATA

1. Relative Nature of 315 Addressing

A 315 address consists of 5 digits. In absolute format, part of the address is stored in the A field of a single stage instruction (also the B field for a double stage instruction). A or B are one slab in length and can contain only 3 digits. The remainder of the address is stored in an index register named in the X (or Y) field, thus making it relative to the A (or B) address.

An index register named in an instruction must contain the appropriate value to augment the contents of the A (or B) field to equal the correct memory address. For programming convenience, the following index registers will be set initially by the executive system to contain the following values:

IR00 through IR09 - 00 000 through 09 000
IR16 through IR25 - 10 000 through 19 000

For CRAM only, IR24 contains 38 000 and IR25 contains 39 000.

2. Static Addresses

The Compiler can make use of these standard register settings when constructing addresses for instructions, constants, and data units that have a fixed location in memory. It will search a table of IR-numbers and their contents to find an index register which contains a value which is within 1000 of the address desired, subtract the register value from the address, and store the difference in the A (or B) field of the machine instruction.

For example, assume that in the allocation of memory, a two-slab constant named AMOUNT was assigned an area in memory beginning at location 03210. The following instruction:

Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
LD				AMOUNT

would be treated by the Compiler as though it had been written:

Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
LD		2	3	210

3. Dynamic Addresses

The standard register settings cannot be used when it is necessary to advance through a series of similar data units within a record, or from one record to the next in a block. In this case a register must be assigned in order not to interfere with the normal pattern of the Compiler for static addresses, as was described above.

For example, assume that index register 10 has been assigned by the programmer to reference PARTSREC and data units within PARTSREC, and that in the allocation of memory, PARTSBLOCK was assigned a 1400 slab area in memory beginning at location 04321. The following instruction:

Op (19-22)	V (23-24)	LENGTH / LEVEL (25-26)	X (27-28)	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE (29-48)
LD				INVENDATE

would be treated by the Compiler as though it had been written:

Op (19-22)	V (23-24)	LENGTH / LEVEL (25-26)	X (27-28)	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE (29-48)
LD		3	10	007

In this case, the programmer would have had to load the base address of the block (04321 which is also the address of the first slab of the first record) into index register 10.

In order to reference INVENDATE in the second record of the block, the program must increase the contents of index register 10 by the size of the record (20).

4. Indexing

This technique of using an index register and an A address to reference repeated data units is called "Indexing".

The assigned index register will contain the base address (first slab) of the record. The values stored in the A (or B) portion of machine instructions that reference the data units within the record will indicate the position of the unit relative to the base address, for example, 000 if the first slab, 001 if the second slab, etc. To reference the second record of the block, add the record size to the contents of the index register. This will set the register to the first slab of the second record. All related machine instructions will still reference the correct data units since the A (or B) values of the instructions will be added to the new value of the register.

After all the records in the block have been processed, a new block of data is read into memory and the base address is loaded into the assigned index register. Thus the register is reset to the address of the first slab of the first record, and the cycle may be repeated for each record in the new block.

For additional details regarding the assignment and use of index registers, see the control instructions DATA, INDEX and REDFN in this chapter, and BASE in Chapter IV.

G. DATA

This control instruction indicates to the Compiler that this line and the following lines with blank Operation codes are Data Definitions. The Compiler will allocate memory for the described data units according to the contents of the location counter.

This control instruction is used to: (1) name the data units, (2) show their physical position within the largest unit and their logical relationship to other data units, (3) show the length and type of the data units, and (4) show the index register assigned to be used for addressing the data units.

PAGE	LINE	REFERENCE	Op	V	LENGTH LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
1 2 3	4 5 6	8 9 10 11 12 13 14 15 16 17	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
			DATA		lvl	reg	length, type

DATA must be entered in the Operation column for the first level 1 in a series. The Operation column of subsequent Data Definition lines may be left blank unless an INDEX or a REDFN entry occurs, in which case DATA is again entered in the first line following the INDEX or REDFN.

The following is an example of a Data Definition of a file. The level 1 entry assigns the name PARTSBLOCK to an input/output area in memory, and states the size of this area is 1400 slabs. The Compiler will reserve 1400 slabs in memory and will associate the Reference PARTSBLOCK with the base address of this area. Index register 10 has been assigned by the programmer to be used to advance through the four QMEQ data units. Note that only the first of the four QMEQ data units need be listed since the remaining three fall within the 20 slabs allocated for the level 2 entry.

NUMBER OF SLABS PER RECORD: 20 — NUMBER OF RECORDS PER BLOCK: 70

REFERENCE	I-REG.	BASE
01 PARTSBLOCK	10 (11)	

FILE NAME: Parts Procurement File — RECORD NAME: Manufactured Parts Record

000	001	002	003	004	005	006	007	008	009	010	011	012	013	014	015	016	017	018	019
02 PARTSREC																			
03 PARTNUM				03 DESIGN DATE			03 INVENDATE			03 STATUS									
04 SERIAL			04 CODE				04 MON	04 DAY		04 IN PROCESS		04 QMEQ		→					
XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX	XX

PAGE	LINE	REFERENCE	Op	V	LENGTH LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
1 2 3	4 5 6	8 9 10 11 12 13 14 15 16 17	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
020	010	PARTSBLOCK	DATA		01		1400S
020	020		INDEX				10, 11
020	030	PARTSREC	DATA		02	10	20S
020	040	PARTNUM			03		4S
020	050	SERIAL			04		3S
020	060	CODE			04		1S
020	070	DESIGNDATE			03		3S
020	080	INVENDATE			03		3S
020	090	MON			04		1S
020	100	DAY			04		1S
020	110	STATUS			03		10S
020	120	INPROCESS			04		2S
020	130	QMEQ			04	11	2S

H. INDEX

This control instruction indicates to the Compiler that the index registers listed will contain the base address of the data unit described on the next line of the programming sheet.

Op (19-22)	V (23-24)	LENGTH / LEVEL (25-26)	X (27-28)	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE (29-48)
INDE	X			IR number(s)

Enter INDEX in the Operation column, and enter the assigned index register numbers, separated by commas, in the Operands column. If more than one index registers are assigned, they must be consecutive.

In the X column of succeeding lines, enter the specific index register number assigned to that data unit. The number need not be repeated if it is the same as that on the previous line.

This index register assignment applies to the data unit named on that line, and for all subunits (lower level units, designated by a level number that is larger than that of the previous unit).

If INDEX were not used, the Compiler would calculate addresses for data units using the standard index register settings. However, where INDEX is used, the Compiler will assume that the base address will be loaded in the registers. The Compiler will calculate the A (or B) field of machine instructions referencing each data unit according to their relative position from the base address.

For example, if each of the data units defined below was referenced by a machine instruction, and the base address was 04321, the Compiler would generate coding for the machine instructions with entries in the X and A fields as shown below.

In order for these addresses to function correctly, the base address must be in the index register. It is the programmer's responsibility to load and advance the index registers; however, many of the macro instructions will do this automatically.

REFERENCE (8-17)	18	Op (19-22)	V (23-24)	LENGTH / LEVEL (25-26)	X (27-28)	INSTRUCTIONS: / DATA DEFINITIONS: (29-35)	X_XFC / Y_yQG	A / B
PARTSBLOCK		DATA		01		1400S		
		INDE	X			10,11		
PARTSREC		DATA		02	10	20S	@	000
PARTNUM				03		4S	@	000
SERIAL				04		3S	@	000
CODE				04		1S	@	003
DESIGNDATE				03		3S	@	004
INVENDATE				03		3S	@	007
MON				04		1S	@	007
DAY				04		1S	@	008
STATUS				03		10S	@	010
INPROCESS				04		2S	@	010
QMEQ				04	11	2S	,	012

I. REDFN

This control instruction indicates to the Compiler that the last previous data unit with the level named will be redefined and the corresponding memory area will be reallocated to the data units described on the following lines.

Op (19 20 21 22)	V (23 24)	LENGTH / LEVEL (25 26)	X (27 28)	INSTRUCTIONS: / DATA DEFINITIONS: (29 30 31 32 33 34 35)
R E D F	N	lvl		

The Compiler will reset the location counter to the address of the previously defined data unit having the same level number and will reassign these addresses beginning with the data unit described on the next line. A DATA instruction must occur immediately after a REDFN entry and before any other instruction except INDEX.

The use of REDFN permits an area in memory to have more than one data definition. For example, an input block may contain several different types of records. The organization of the data units may vary depending upon the particular type of record.

In our example, some of the manufactured parts may have been discontinued in which case the last ten slabs of the record would be redefined. Note that slabs 17, 18, and 19, which do not contain any useful information, are ignored in the redefinition since they fall within the 20 slabs allocated for the level 2 entry (PARTSREC).

NUMBER OF SLABS PER RECORD: 20 NUMBER OF RECORDS PER BLOCK: 70

REFERENCE	r-REG.	BASE
01 PARTSBLOCK	10	

1 1

FILE NAME: Parts Procurement File RECORD NAME: Manufactured Parts Record

000	001	002	003	004	005	006	007	008	009	010	011	012	013	014	015	016	017	018	019
02 PARTSREC																			
03 PARTNUM				03 DESIGN DATE			03 INVENDATE			03 DISCONDATE			03 AUTHNUM		03 ON HAND				
04 SERIAL			04 CODE				04 MON	04 DAY		04 MO	04 DA	04 YR							
X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X	X X			

PAGE	LINE	REFERENCE	Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
1 2 3	4 5 6	8 9 10 11 12 13 14 15 16 17	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
020	010	PARTSBLOCK	DATA		01		1400S
020	020		INDEX				10, 11
020	030	PARTSREC	DATA		02	10	20S
020	040	PARTNUM			03		4S
020	050	SERIAL			04		3S
020	060	CODE			04		1S
020	070	DESIGNDATE			03		3S
020	080	INVENDATE			03		3S
020	090	MON			04		1S
020	100	DAY			04		1S
020	110	STATUS			03		10S
020	120	INPROCESS			04		2S
020	130	QMEQ			04	11	2S
020	140		REDFN		03		
020	150	DISCONDATE	DATA		03		3S
020	160	MO			04		1S
020	170	DA			04		1S
020	180	YR			04		1S
020	190	AUTHNUM			03		2S
020	200	ONHAND			03		2S

Data may be redefined at any level, including level 1 and level 2. The following is an example of a redefined record, level 2.

NUMBER OF SLABS PER RECORD: 20 NUMBER OF RECORDS PER BLOCK: 70

REFERENCE	I-REG.	BASE
01 PARTSBLOCK	10 11	

FILE NAME Parts Procurement File RECORD NAME Purchased Parts Record

02 PARTSRECP																			
03 PARTNUMP				03 LASTPURCH			03 ON ORDER		03 VALUEYTD			03 QPREQ		→					
04 SERIALP			04 CODEP	04 M	04 D	04 Y													
XX	XX	XX	XX	XX	XX	XX	XX	XX	XXX	XXX	XXX	XX	XX	XX	XX	XX	XX	XX	XX
000	001	002	003	004	005	006	007	008	009	010	011	012	013	014	015	016	017	018	019

PAGE	LINE	REFERENCE	Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
1 2 3	4 5 6	8 9 10 11 12 13 14 15 16 17	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46
020	010	PARTSBLOCK	DATA		01		1400S
020	020		INDEX				10, 11
020	0[illegible]	PARTSRE[illegible]	DATA				20S

PAGE	LINE	REFERENCE	Op	V	LENGTH / LEVEL	X	OPERANDS
[illegible]	190	AU[illegible]UM			03		[illegible]
020	200	ONHAND			03		2S
020	210		REDFN		02		
020	220	PARTSRECP	DATA		02	10	20S
020	230	PARTNUMP			03		4S
020	240	SERIALP			04		3S
020	250	CODEP			04		1S
020	260	LASTPURCH			03		3S
020	270	M			04		1S
020	280	D			04		1S
020	290	Y			04		1S
020	300	ONORDER			03		2S
020	310	VALUEYTD			03		7.2
020	320	QPREC			03		2S

J. CRAM TRACK LABELS

When using CRAM files, the installation deck number, the CRAM card number, and the track number for the file must be contained in the first four slabs of each block. This four-slab data unit must be shown in the data definitions for the file. Show this unit as the first level 2 entry. It is not necessary to assign a name in the Reference column to the track label.

In the following illustration, note that the INDEX entry is placed after the four slabs for the track label. Thus all associated index registers will be set up properly to the beginning of the first record, immediately following the label. Note also that the four slabs are included in the total block size shown for the level 1 entry.

PAGE	LINE	REFERENCE	Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
1 2 3	4 5 6	8 9 10 11 12 13 14 15 16 17	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
020	010	PARTSBLOCK	DATA		01		1404S
020	020				02		4S
020	030		INDEX				10, 11
020	040	PARTSREC	DATA		02		20S
020	050	PARTNUM			03		4S
020	060	SERIAL			04		3S
020	070	CODE			04		1S

K. VARIABLE LENGTH RECORDS

1. CRAM

When using variable length CRAM records, the record length must be contained in the first slab of each record. This one-slab data unit must be shown in the data definitions for the file. Show this unit as the first level 3 entry. It is not necessary to assign a name in the Reference column to the record length.

In the following illustration, note that the INDEX entry is placed after the four slabs for the track label, but before the record definition (level 2 entry) which contains the one-slab unit for the record length. Note also that the slab is included in the record size, and is included in the total block size shown for the level 1 entry.

PAGE	LINE	REFERENCE	Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
1 2 3	4 5 6	8 9 10 11 12 13 14 15 16 17	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
020	010	PARTSBLOCK	DATA		01		1474S
020	020				02		4S
020	030		INDEX				10,11
020	040	PARTSREC	DATA		02	10	21S
020	050	RL			03		1S
020	060	PARTNUM			03		4S
020	070	SERIAL			04		3S
020	080	CODE			04		1S

2. Magnetic Tape

When using variable length magnetic tape records, the record length must be contained in the first slab of each record. This one-slab data unit must be shown in the data definitions for the file. Show this unit as the first level 3 entry. It is not necessary to assign a name in the Reference column to the record length.

In the following illustration, note that the INDEX entry is placed before the record definition (level 2 entry) which contains the one-slab unit for the record length. Note also that the slab is included in the record size, and is included in the total block size shown for the level 1 entry.

PAGE	LINE	REFERENCE	Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
1 2 3	4 5 6	8 9 10 11 12 13 14 15 16 17	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
020	010	PARTSBLOCK	DATA		01		1470S
020	020		INDEX				10,11
020	030	PARTSREC	DATA		02		21S
020	040	RL			03		1S
020	050	PARTNUM			03		4S
020	060	SERIAL			04		3S
020	070	CODE			04		1S

IV. COMPILER CONTROL INSTRUCTIONS

A. GENERAL

Control instructions provide a number of functions peculiar to the compiling process. Among these are: to set up constants, to direct the allocation of memory and assignment of addresses, to define data and assign index registers, to control the Compiler output, and to identify special source program information.

Where appropriate, references may be used as operands of some of the control instructions. Where a reference is used as an operand of the control instructions ORIGIN, OVRLAY, SAVE or EQUATE, it must have been defined previously in the source program. In all other cases, the reference may be defined anywhere in the source program.

Absolute addresses are permitted as operands; however, be very careful in their use.

A control instruction must be complete on one line. The operand may not be extended to the next line.

Remarks may be entered in most cases. However, where remarks are not permitted, this is indicated in the description of the particular instruction.

Always be sure to leave at least two spaces before beginning the remarks.

Following is a complete list of the Compiler control instructions and a description of their use.

B. DESCRIPTION OF CONTROL INSTRUCTIONS

1. For Definition of Constants

There are two methods by which constants may be set up. A constant may be written as a literal in the operand of an instruction, in which case the Compiler will store it as an out-of-line constant in the Program Safe Area and store its address as the A or B operand of the instruction. Use of literals as operands is described later in this chapter.

Constants may be stored in-line by use of various control instructions. In this case, the constant will be stored in memory where it occurs in the source program. The constant is assigned an address equal to the contents of the location counter (next available slab). The location counter is then advanced by the size of the constant (number of slabs--usually specified in the Length column), and the location counter is thus set to the next available slab following the constant just set up.

A name shall be assigned to each constant (entered in the Reference column for that instruction) so that it may then be used as the operand of other instructions. The Compiler will store the address of the constant as the A or B operand of the compiled instruction, and where appropriate, will store the length (as L-1) in the F field of the instruction in absolute format.

The following control instructions define in-line constants having the desired format, and containing the characters (or values) specified in the operands column.

a. ALPHA

This control instruction will set up a constant in Alpha form (two 6-bit characters per slab).

REFERENCE (8–17)	18	Op (19–22)	V (23–24)	LENGTH / LEVEL (25–26)	X (27–28)	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE (29–48)
name		ALPH	A	len		Alpha characters

Successive pairs of characters, beginning at the first position of the Operands column, are placed in successive slabs in memory. The Alpha characters are left justified and the field is filled out to the right with spaces.

The following entry:

REFERENCE (8–17)	18	Op (19–22)	V (23–24)	LENGTH / LEVEL (25–26)	X (27–28)	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE (29–48)
REF1		ALPH	A	3		ABCD5

becomes:

AB	CD	5 Ø

Any of the 64 characters in the 315 Code Chart may be used.

The Length column specifies the length of the constant (number of slabs to be set up). Maximum length is 21 slabs.

Note: For this instruction, a space does not terminate the operand. The number of characters to be included in the operand is specified by the number of slabs entered in the Length column. If remarks are entered, be sure to allow for the correct number of characters in the constant so that a portion of the remarks will not be picked up and stored as part of the constant.

ALPHA may be used to set up constants in Alpha form.

ALPHA may also be used to include an area of slabs containing spaces in an overlay which can be used later to make program patches through the Librarian.

b. DIGIT

This control instruction will set up a constant in Digit form (three 4-bit characters per slab).

REFERENCE (8–17)	18	Op (19–22)	V (23–24)	LENGTH / LEVEL (25–26)	X (27–28)	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE (29–48)
name		DIGI	T	len		Digit characters

Successive triples of characters, beginning at the first position of the Operands column, are placed in successive slabs in memory. The Digit characters are left justified and the field is filled out to the right with spaces.

The following entry:

Reference	Op	V	Length	X	Operands
REF2	DIGIT		2		12345

becomes:

1 2 3	4 5 ⌀

Only the Digit characters (the 16 characters appearing in the first row of the 315 Code Chart) may be used, that is:

0 through 9 @ , ⌀ & . -

If any non-digit characters are used, the Compiler will retain only the right-hand four bits, and will print an error notation on that line of the listing.

The Length column specifies the length of the constant (number of slabs to set up). Maximum length is 14 slabs.

Note: For this instruction, a space does not terminate the operand. The number of characters to be included in the operand is specified by the number entered in the Length column. If remarks are entered, be sure to allow for the correct number of characters so that a portion of the remarks will not be picked up and stored as part of the constant.

DIGIT may be used to set up constants in Digit form.

DIGIT may also be used to include an area of slabs containing zeros in an overlay which can be used later to make program patches through the Librarian.

c. NUMBER

This control instruction will set up a constant in numeric form.

REFERENCE		Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
name		NUMBER		len		numerals

Successive triples of characters, beginning at the first position of the Operands column, are placed in successive slabs in memory. The Digit characters are right justified and the field is filled out to the left with zeros.

The following entries:

Reference	Op	Length	X	Operands
REF3	NUMBER	2		12345
REF4	NUMBER	2		-4321
REF5	NUMBER	3		123.4567

become: | 0 1 2 | 3 4 5 | | - 0 4 | 3 2 1 | | 0 0 1 | 2 3 4 | 5 6 7 |

Only the decimal numerals zero through nine, the minus sign, and the decimal point (0 through 9 - .) are permitted. If any other characters are used, the Compiler will retain only the right-hand four bits, and will print an error notation on that line of the listing. If a minus sign is entered as the first character in the Operands column, it will be stored in the left-hand Digit position of the field. If a decimal point is included, it will not occupy any memory space in the constant. However, the position of the decimal point will be noted by the Compiler so that it may be used by macro instructions written to handle a decimal point in a numeric constant.

The Length column specifies the length of the constant (number of slabs to be set up). Maximum length is 8 slabs.

Note: For this instruction, a space does terminate the operand. If the number of characters in the operand is greater than the length specified, the Compiler will pick up only the number specified beginning at the first position of the Operands column.

NUMBER may be used to set up constants in numeric form which will be used for arithmetic operations.

d. PAIR

This control instruction will set up a two-slab address-constant.

REFERENCE	Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
name of address-	PAIR				address to be stored in two slabs
constant					

The operand is evaluated and that value (which represents a five-digit address) is placed right justified in a memory pair (a two-slab field to be used as an address-constant). The left-hand six bits of each pair are normally set to zero by the Compiler. The value of the operand may range from -39,999 to + 39,999.

The operand may be a reference or may be an absolute address.

The operand may be negative, and if so, the minus sign will be stored as the left-hand four bits of the pair. However, as this is intended as an address-constant, the Compiler will flag the line as a possible error.

If the next available slab is memory location 01112; and the address of AMOUNT is 03210, the following entries:

R E F 6	P A I R			A M O U N T
R E F 7	P A I R			1 2 3 4

become:

01112	01113		01114	01115
0 0 3	2 1 0		0 0 1	2 3 4

REF6: 01112

REF7: 01114

PAIR may be used to set up individual address-constants for indexing or for index register manipulation, and may also be used to set up a series of addresses, as in a table.

Since only the right-hand 18 bits are required to represent any address on the 315 Computer, the unused portion (left-hand six bits) may be utilized as a memory flag. This can be done by writing an A (for Alpha) or a D (for Digit) left justified in the X column (in column 27) followed by the desired character or digit (in column 28).

If an A is entered (in column 27), the Compiler will place the desired character (entered in column 28) in the left-hand six bits of the pair. To obtain the character "space", enter "A Ø " in the X column.

If a D is entered (in column 27), the Compiler will place the right-hand four bits of the desired digit in the left-hand four bits of the pair, with the remaining two bits being set to zero. If a D is entered and the next character has a one in either of the zone bits, the Compiler will retain only the right-hand four bits and will print an error notation on that line of the listing.

If the operand is negative, the left-hand six bits of the pair should not be used to store a flag since the generated character will interfere with the minus sign. Also, if an A is entered and the next character is ' (Binary 111100), [(111101),] (111110), or \ (111111), the entire field will appear negative.

If the X column is left blank, all six bits will be set to zero.

The following are examples of how memory pairs may be used:

To load the address 012345 in IR27:

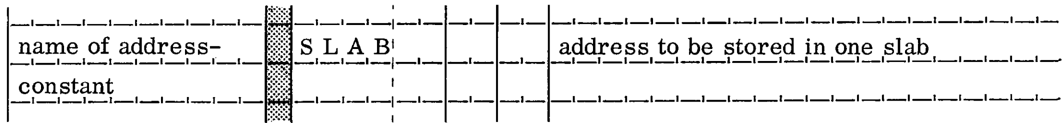
P A I R A D D R	P A I R			1 2 3 4 5
	L D R			P A I R A D D R , 1 , 2 7

Assuming a block will be read in from magnetic tape, to load the base address of the first record in PARTSBLOCK:

I N P U T	P A I R			P A R T S B L O C K
	S L D R			I N P U T , 2 , 1 0

e. SLAB

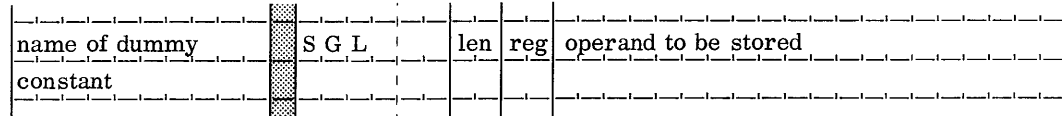
This control instruction will set up a one-slab address-constant.



The operand is evaluated and that value (which represents a three-digit address) is placed in a slab in memory. If the value is greater than 999, the Compiler will retain only the right-hand 12 bits and will print an error notation on that line of the listing.

f. SGL (Single)

This control instruction will set up a dummy single-stage instruction.



The Length, X, and Operands columns are evaluated and the Compiler will convert these entries into absolute instruction format and store the result in a two-slab field as a single-stage instruction with an operation code of zero.

SGL may be used to set up dummy instructions which can be used for instruction modification using the machine instruction Binary Add to Accumulator.

2. Literal Constants

Constants can be set up in a program by using control instructions. The constants can be named and described, and the Compiler will set up the corresponding values in memory.

Constants can also be set up in a program by simply including their literal values as the operands of the instructions which refer to these values. This may be done by writing in the Operands column the number sign and a letter designating the type of literal, followed by the actual value enclosed in parentheses. For example:

Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE	REMARKS / REMARKS
19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48	49 50 51 52 53 54 55 56 57 58 59
T Y P E	A			# A (A B C D E)	

The Compiler will automatically allocate memory space for the information represented by the literal. One-slab literals will be set up as the operand of the instruction in absolute format if the instruction is one which permits this. Literals larger than one slab will be set up as a constant in the Program Safe Area and the address of this literal constant will be stored as the operand of the instruction.

The Compiler will set up a literal in the Program Safe Area only once, even though the same literal may be entered as the operand in several instructions. Duplicate literals will not result in duplicate constants, so that memory space is not wasted.

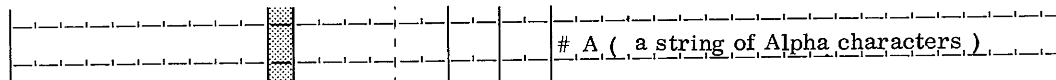
The length of a literal is limited by the size of the operand which the particular instruction can accommodate. A literal cannot extend to the next line of coding. For example, a literal used with the machine instruction LD could not be larger than

eight slabs (maximum length of the accumulator), while a literal used with the machine instruction TYPE:A could not be larger than 21 slabs (42 columns remaining in the Operands column of one line of the programming worksheet following the #A(   ) entry).

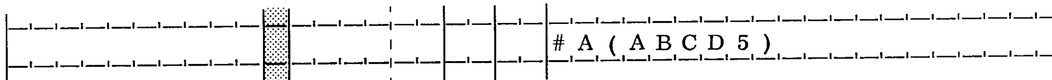
There are four types of literals: Alpha, Digit, Numeric, and Reference.

a. ALPHA literal

Alpha literals are written:



This literal is set up in exactly the same manner as in the control instruction ALPHA. The characters enclosed by the parentheses will be left justified in the correct number of slabs, two 6-bit characters per slab, and the field will be filled out to the right with spaces.



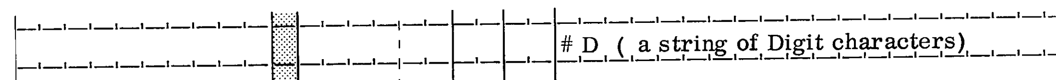
A B	C D	5 Ø

Any of the 64 characters in the 315 Code Chart, except the right parenthesis, may be used.

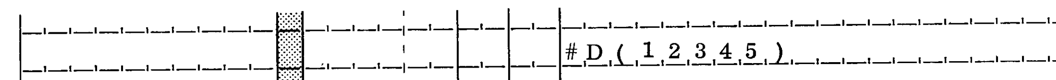
The right parenthesis ")" cannot appear within a literal. If parentheses are required as characters in a constant, the constant must be set up in the program using the control instruction ALPHA.

b. DIGIT literal

Digit literals are written:



This literal is set up in exactly the same manner as in the control instruction DIGIT. The Digits enclosed by the parentheses will be left justified in the correct number of slabs, three 4-bit characters per slab, and the field will be filled out to the right with spaces.



1 2 3	4 5 Ø

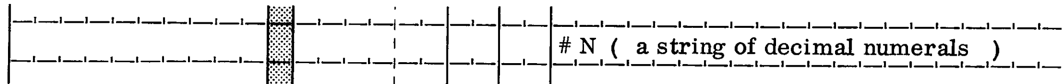
Only the Digit characters (the 16 characters appearing in the first row of the 315 Code Chart) may be used, that is:

0 through 9 @ , ⌀ & . -

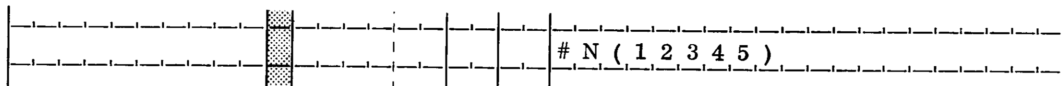
If any non-digit characters are used, the Compiler will retain only the right-hand four bits, and will print an error notation on that line of the listing.

c. Numeric literal

Numeric literals are written:



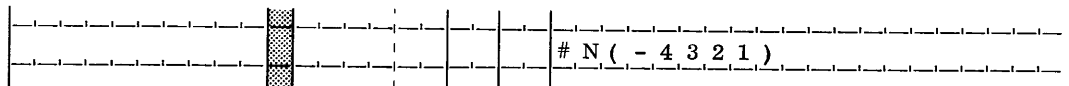
This literal is set up in exactly the same manner as in the control instruction NUMBER. The Digit characters are right justified in the correct number of slabs, three 4-bit characters per slab, and the field will be filled out to the left with zeros.



0 1 2	3 4 5

Only the decimal numerals zero through nine, the minus sign, and the decimal point (0 through 9 - .) are permitted. If any other characters are used, the Compiler will retain only the right-hand four bits, and will print an error notation on that line of the listing.

If a minus sign is entered as the first character of the literal, it will be stored in the left-hand Digit position of the field.



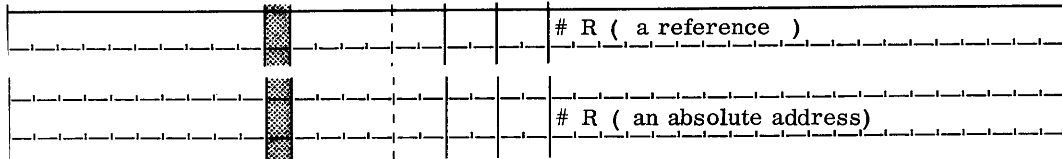
- 0 4	3 2 1

If a decimal point is included, it will not occupy any memory space in the constant. However, the position of the decimal point will be noted by the Compiler so that it may be used by macro instructions written to handle a decimal point in a numeric literal.

0 0 1	2 3 ^ 4	5 6 7

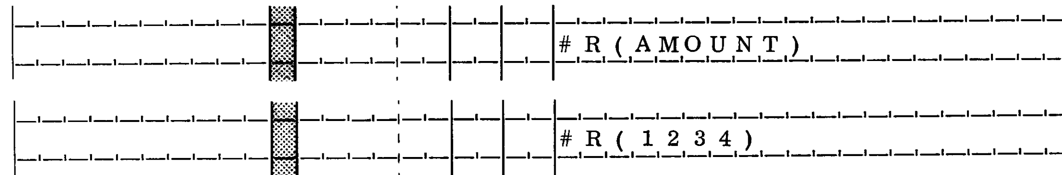
d. Reference literal

Reference literals are written:

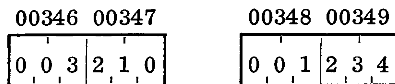


This literal is set up in exactly the same manner as in the control instruction PAIR. The entry enclosed by the parentheses is evaluated and that value (which represents a five-digit address) is placed right justified in a memory pair (a two-slab field to be used as an address-constant). The field will be filled out to the left with zeros. (Note: The left-hand six bits cannot be used as a memory flag.)

If the next available slab in the Program Safe Area is memory location 00346 and the address of AMOUNT is 03210, the following entries:



become:

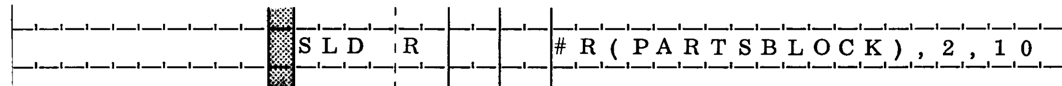


Note: Compound references may not be used in a reference literal. If a compound reference is desired, use the control instruction PAIR as described earlier in this chapter.

The following are examples of how reference literals may be used in lieu of the control instruction PAIR:

To load the address 12345 in IR27:

Assuming a block will be read in from magnetic tape, to load the base address of the first record in PARTSBLOCK:



3. To Control the Location Counter

The Compiler will allocate memory according to the needs of the entire system, that is, the requirements of the object program, the executive routines, and the Librarian. Several of these memory requirements are fixed and constant for all programs; others

are variable. Starting in the low order of memory, areas are reserved for Universal Temporary Storage, the Universal Safe Area, and File Table O, which are used by the executive routines and by the Librarian. Next are the object program file tables, followed by the Program Safe Area, which contains out-of-line coding such as generated literal constants, generated macro subroutines, and, usually, some executive routines. The in-line coding portion of the object program (containing the program instructions, input-output areas, constants, ect., generated in-line) follows the Program Safe Area. (Refer to the CRMX-II or STEP Manual for absolute address memory layout.)

In allocating memory for the in-line coding portion of the object program, the Compiler will start with the first available slab following the Program Safe Area, and assign this location to the first element of the object program that will occupy memory. This may be an instruction, a constant or data unit, or a work area to be used for input, output, or temporary storage, etc.

As each assignment is made, a count is maintained of the amount of memory allocated (length in slabs of the size of the unit). For example, it will add two (slabs) if a single stage instruction is generated, or four for a double stage instruction. This counting process is done by the Compiler in the "location counter". At any point in the Compiler run, the contents, or value, of the location counter will equal the address of the next available location, that is, the address to be assigned to the next unit.

Normally, the location counter is set initially to the first available slab following the Program Safe Area. As each assignment is made, the number of slabs assigned (length of the unit) is added to the contents of the location counter. The result is the address of the next instruction or memory area.

It is possible to modify this normal counting procedure. The control instructions ORIGIN, OVRLAY, SAVE and LITORG provide this ability. Care should be exercised in the use of absolute addresses to reset the location counter.

a. ORIGIN

This control instruction causes a new program block to be output beginning with the next instruction of the source program.

REFERENCE		Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
		ORIG	IN			desired value of location counter

The operand is evaluated and the location counter is reset to this value.

The value of the operand must lie in the range 0 to 39,999.

It is advisable to use a reference (rather than an absolute value) as the operand of an ORIGIN.

A reference used as an operand must have been previously defined.

If a name is entered in the Reference column for this instruction, it will be assigned the resulting value of the operand evaluation.

A new block of coding is output beginning with the instruction following the ORIGIN control instruction. The new contents of the location counter (value of the operand) will be assigned as the address of the first slab of the new block.

In generating the object program, the Compiler will itself make use of the control instruction ORIGIN.

Normally the Compiler will assign memory addresses to the generated in-line coding beginning at the memory area immediately following the Program Safe Area. By using ORIGIN (and an operand) as the first instruction of the program, a different memory location can be specified where the coding will start.

b. OVRLAY

In some instances, especially where a limited memory size is a problem, it may be desirable to bring a part of the program into main memory at locations allocated to a previous portion of the program. This is known as an overlay and is accomplished by using the control instruction OVRLAY. This control instruction causes a new overlay to be created beginning with the next instruction of the source program.

REFERENCE		Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
overlay name		OVRL	AY			putaway address

The operand is evaluated and the location counter is reset to this value.

The value of the operand must be in the range 0 to 39,999.

A reference used as an operand must have been previously defined.

A unique name must be assigned to each overlay (entered in the Reference column of this instruction). If a name is entered in the Reference column, it will be assigned the number of the new overlay (e.g. 01, 02, etc.), not the value of the putaway address. This assigned overlay number is used when making program patches during the Librarian run.

A new block of coding is output beginning with the instruction following the OVRLAY control instruction. This block becomes the first block of a new overlay. The new contents of the location counter (value of the operand) will be assigned as the address of the first slab of the overlay.

The main program is automatically set up by the Compiler as though it were Overlay 00. When the object program is run, the System Supervisor will read this portion of the program into memory.

Any subsequent overlays must be created by the use of OVRLAY. Each new overlay will be assigned the next consecutive overlay number (01, 02, etc.).

In producing overlays, the Compiler will also generate the control information necessary to permit it to be read into its correct position in memory. To call an overlay into memory, use the control instruction CALLC, or CALL, as described in the Macro Instructions Manual. Note that Overlay 00 is intended to contain the main program and cannot be recalled into memory.

In assigning a putaway address for an overlay, be certain that the overlay when called into memory will not occupy an area that contains data needed by the portion of the program currently being processed.

c. SAVE

This control instruction causes an area in memory to be left unaffected, by advancing the contents of the location counter.

REFERENCE		Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
		SAVE				number of slabs

SAVE merely advances the location counter by the value of the operand. It does not produce any coding in the object program.

The value of the operand must be positive or zero, and must be in the range 0 to 39,999.

If a Reference is used as an operand, it must have been previously defined.

If a name is entered in the Reference column for this instruction, it will be assigned the current value of the location counter, that is, the value before addition.

A new block of coding will be generated beginning with the instruction following the SAVE control instruction. When the object program is read into memory, no coding will be read into the area skipped by this SAVE. In this way, SAVE can be used to retain information left in memory from a previous overlay.

Note: This "saved" memory area cannot be used to make program changes (using patches) through the Librarian. To reserve memory for patches by the Librarian, use the control instruction ALPHA or DIGIT to include an area of slabs containing spaces or zeros, etc., or use the macro instruction ZERO.

SAVE can also be used to reserve an area in memory which can be used for input or output, or as a work area.

SAVE can be used as a limiter to mark the end of the memory area immediately preceding the SAVE control instruction. SAVE with an operand of zero (0) will not advance the location counter. A name entered in the Reference column for this instruction will be assigned the same value as the location counter; however, if this reference is used as the operand of an instruction, the absolute coding generated by the Compiler will contain only the starting location of the saved area, but not the length.

d. LITORG

This control instruction causes the memory location of the Program Safe Area to be changed.

REFERENCE		Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
		LITO	RG			absolute address of first
						slab in Program Safe Area

The Program Safe Area will be relocated to start at the address assigned to the operand.

The operand must be an absolute address and must be in the range 0 to 39,999. A reference may not be used as an operand of this instruction.

The Program Safe Area is a variable length area containing the literal constants generated out-of-line by the Compiler, macro subroutines generated out-of-line, the Rescue Routine, and for magnetic tape programs, the Extremity Routine. Normally the Compiler will store the Program Safe Area in memory immediately following the file tables. This position in memory can be changed by using LITORG.

LITORG may be used only once in a program.

4. For Referencing

In many cases it is useful to refer to the same constant or instruction or data area with several names. It is also useful to assign the same values or addresses to several names used previously. The following control instruction will provide this ability.

EQUATE

This control instruction will associate the value, address, or reference, entered as the operand, with the name entered in the Reference column. Only one operand is permitted for each EQUATE instruction. All descriptive information previously assigned to the operand is assigned to the reference to which it is equated (i.e. length, associated index registers, type).

REFERENCE		Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
name assigned		EQUA	TE			value or address or reference

The operand is evaluated and the resulting value, or address, is associated with the name entered in the Reference column.

The operand may be a single value, address, or reference, or it may be compound.

The names used in the Reference column and the references used in the Operands column may be any allowable name (e.g., may refer to a constant, an instruction, a data unit, etc.).

A reference used as an operand must have been previously defined.

Examples:

REFERENCE		Op	V	LENGTH / LEVEL	X	OPERANDS
MAXIMUM		EQUA	TE			999
LOCATION7		EQUA	TE			12345
LIMIT		EQUA	TE			UNIT+3

EQUATE may be used to:

assign a name to numeric value or address

assign a numeric value or address to a name,

associate a name with a name that has been previously defined.

assign the same value or address to two or more names.

EQUATE can be used to assign symbolic names to index registers or to jump registers. Enter the register number in the Operands column and assign a one- or two-character symbol in the Reference column. For example:

REFERENCE (8–17)	18	Op (19–22)	V (23–24)	LENGTH / LEVEL (25–26)	X (27–28)	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE (29–48)
A 1		E Q U A	T E			1 2

The Reference A1 can now be used as an operand of machine instructions which refer to IR12 or to JR12.

Note: Symbolic index register references may not be used in Data Definitions.

A compound operand may contain the operators *,/,+, or - (representing multiplied by, divided by, plus, minus). The Compiler will perform the arithmetic in the following order: multiplication, division, addition, subtraction. All arithmetic is strictly digital; the fraction part of each computation is ignored.

5. For Program Addressing

a. BASE

In constructing addresses, the Compiler must be able to depend on the values contained in certain index registers at the time the object program is executed. For programming convenience, the following index registers will automatically be set initially by the CRMX-II or STEP System Supervisor to contain the following values:

IR00 through IR09 - 00 000 through 09 000
IR16 through IR25 - 10 000 through 19 000

For CRAM only, IR24 contains 38 000 and IR 25 contains 39 000

The Compiler can make use of these standard register settings when constructing addresses for instructions, constants, and data units that have a fixed location in memory. It will search a table of IR-numbers and their contents to find a register which contains a value which is within 1000 of the address desired, subtract the register value from the address, and store the difference in the A (or B) field of the machine instruction.

When it becomes necessary to change a standard setting, the Compiler must be informed of the new value so that the assembled instructions and the Compiler printout will contain the correct addresses.

The control instruction BASE informs the Compiler that while the instructions that follow are being assembled, it can assume that the index registers shown in the Operands column will contain the values shown.

REFERENCE (8–17)	18	Op (19–22)	V (23–24)	LENGTH / LEVEL (25–26)	X (27–28)	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE (29–48)
		B A S E				$register_1$ /$value_1$, $register_2$ /$value_2$ . . .

The value may be absolute, or it may be a reference. The value may not be compound, and literals may not be used.

These stated values must then be loaded by the programmer in the respective registers (using the machine instruction LD:R or the macro instruction MLDR).

BASE guarantees to the Compiler that during the execution of the instructions which follow, certain values will be in certain index registers. These guarantees hold until changed by a subsequent BASE. (The new values must also be loaded by the programmer into the index registers.)

A situation where it will be necessary to use BASE would occur in the case of an object program to be run using a computer with 10,000 slabs of memory. Index registers 16 through 25 would then be available for indexing of dynamic addresses (but not IR24 and IR25 for CRAM).

Another example would be the case of an object program to be run using a computer with 40,000 slabs of memory, where it is necessary for the Compiler to construct static addresses for instructions, constants, or data units that have a fixed location in memory which fall in the range greater than 19,999. One solution would be to assign an index register for each 1000 segment affected. If it becomes necessary to use any of the standard registers, BASE must be used to inform the Compiler of the change in contents of the registers. The registers must also be loaded with the correct values at the appropriate point in the program. In this case, it may also be necessary to change the contents back to the initial standard settings if the program jumps to a subroutine stored in a range lower than 19,999.

Remember that the function of BASE is to state index registers and their intended values. The values must still be loaded into the respective registers by the programmer.

b. JVAL

This control instruction states the values which the CRMX-II or STEP System Supervisor will place in the specified jump registers.

REFERENCE		Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
		J V A L				$register_1/value_1$, $register_2/value_2$ . . .

The value may be absolute, or it may be a reference. The value may not be compound, and literals may not be used.

The registers may be any jump registers except 0-5 in tape systems and 6 to 10 in CRAM.

The Compiler will place the values in the Program Header. When the object program is read into memory, the System Supervisor will load these values from the Header into the specified jump registers. The jump registers not mentioned in a JVAL instruction will be set to their standard values.

Only one JVAL instruction may occur in a program.

For a complete list of the standard index and jump register settings, see the CRMX-II and STEP manuals.

6. To Modify the Printout

a. PAGE

Listing on the printer is spaced to the top of the next page. No operand is neces-

sary. This line is not printed.

REFERENCE		Op	V	LENGTH LEVEL	X	INSTRUCTIONS: OPERANDS DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
		P A G E				

b. UNLIST

Listing on the printer is suppressed beginning at the point where this instruction occurs in the source program. No operand is necessary. This line is not printed.

REFERENCE		Op	V	LENGTH LEVEL	X	INSTRUCTIONS: OPERANDS DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
		U N L I	S T			

c. LIST

Listing on the printer is resumed. This instruction counteracts the effect of UNLIST. LIST is the normal case and is assumed at the beginning of the program. No operand is necessary. This line is not printed.

REFERENCE		Op	V	LENGTH LEVEL	X	INSTRUCTIONS: OPERANDS DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
		L I S T				

7. To Delete a Line During Initial Compilation

XXX

This control instruction directs the Compiler to delete from the source program, the instruction with the page-line number indicated.

No operand is necessary. This instruction may be used only during the initial compilation, and only when the Sort option has been specified.

PAGE	LINE		REFERENCE		Op	V	LENGTH LEVEL	X	INSTRUCTIONS: OPERANDS DATA DEFINITIONS: LENGTH, TYPE
1 2 3	4 5 6	7	8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41
page	line				X X X				

Note: The Compiler will accept XXX during a recompilation to delete source lines entered as changes. This could be used to make a correction to a line entered as a change.

8. To Delete One or More Lines During a Recompilation

OMIT

This control instruction directs the Compiler to delete from the source program, the lines with the page-line number(s) indicated. Remarks may be punched, but will not be printed.

To delete a single line, enter its page-line number in the Page-Line column and leave the Operands column blank. Remarks must not appear within the first six character positions of the Operands column.

PAGE	LINE		REFERENCE		Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
1 2 3	4 5 6	7	8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41
page-	-line				OMIT				ØØØØØØ

To delete several consecutive lines, enter the page-line number of the first line to be deleted, in the Page-Line column, and enter the page-line number of the last line to be deleted, in the Operands column.

PAGE	LINE		REFERENCE		Op	V	LENGTH / LEVEL	X	OPERANDS
first	page-				OMIT				last page-line
line									

Note: OMIT should be used only during a recompilation from an ACRAM or an ATAPE. It can be used to omit single lines only in an initial compilation, but XXX is recommended for this purpose.

9. For Definition of Data

   a. DATA

   b. INDEX

   c. REDFN

   These control instructions are described in detail in Chapter III.

10. To Identify Object Program File Specifications

    a. FORMAT C

    This control instruction, preprinted on the CRAM File Specification Worksheet, identifies the file specifications if the object program is to be run using CRAM.

19							
F	O	R	M	A	T		C

    b. FORMAT T

    This control instruction, preprinted on the Magnetic Tape File Specification Worksheet, identifies the file specifications if the object program is to be run using magnetic tape.

19							
F	O	R	M	A	T		T

11. To Identify Compiler Input and Output Specifications

    NEAT

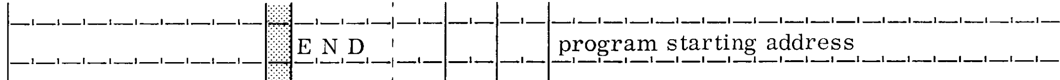
    This control instruction, preprinted on the Control Worksheet, identifies the input and output specifications for this Compiler run. It must be the first line physically input to the Compiler run.

19			
N	E	A	T

12. To Identify the Last Line Input to the Compiler Run

END

This control instruction identifies the last line of coding (end) of the source program. It is the last line physically input to the Compiler run. If the Sort option is specified, it must also bear the highest page-line number. END stops the Compiler input and indicates the Program Starting Address (address of the first instruction to be executed in the object program). An END instruction must be present in every program.



13. To Identify the Last Card of a Correction Deck Input to a Recompilation

Since the end of a correction deck would probably not occur simultaneously with the end of the source program, END would be an ambiguous way to terminate the correction deck. Thus ENDMOD is used to signify the end of the correction deck.

REFERENCE		Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
		E N D M	O D			program starting address

C. ADDRESSING

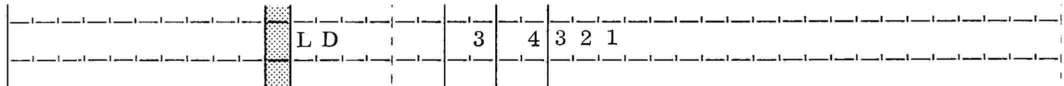
1. Types of Addresses

a. Simple address

A simple address is expressed as a single operand, and may be an absolute address or may be symbolic.

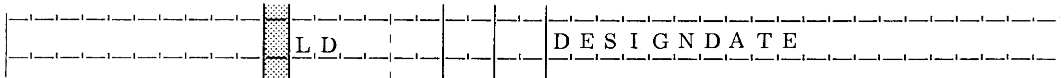
1) Absolute

An absolute address is represented by an actual value entered in the Operands column. For example:



2) Symbolic

A symbolic address is represented by the name (reference) assigned to the particular data unit or memory location being referenced. For example:



Any name that appears in the Reference column in a program may be used as a symbolic operand. The Compiler will convert the operand to the correct absolute addressing notation, which in this case would be 04325 (assuming that the base address of the Manufactured Parts Record is 04321).

b. Compound address

A compound address is represented by any combination of references and absolute values connected by the arithmetic operators: + or - (plus or minus). For example, the operand DESIGNDATE+2 would reference the memory location 04327.

A compound operand must be complete on one line.

Parentheses and literals are not permitted.

The arithmetic operators: * and / (multiplied by and divided by) may be used in a compound operand but are permitted only with the control instruction EQUATE. The Compiler will perform the arithmetic in the following order: multiplication, division, addition, subtraction.

c. Asterisk address

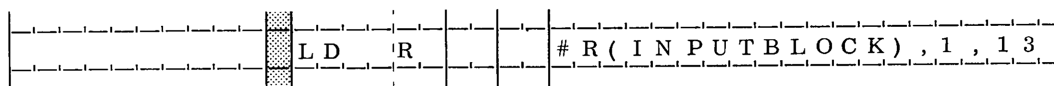
An asterisk address is represented by the symbol * entered as the first character of an operand (first character in the Operands column, first character following a comma, or first character following an arithmetic operator). In evaluating the operand, the Compiler will treat the asterisk as though it were a reference to the instruction itself and will assign a value to the asterisk that is equal to the address of the first slab of coding which results from that line on the programming sheet.

An asterisk may be used as a simple operand. It may be used as part of a compound operand only in EQUATE.

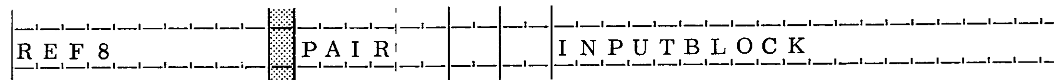
Note: Two additional uses of the asterisk are: (1) to extend remarks to a succeeding line, and (2) to represent the arithmetic operator multiply.

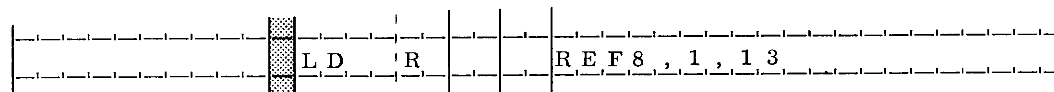
d. Reference literal address

A reference literal is represented as an operand by writing the characters #R followed by a reference enclosed in parentheses. The Compiler will treat this operand as the reference of a two-slab data unit containing the address of the data unit to be referenced. For example, the instruction:



would be interpreted by the Compiler as though it had been written:





For additional details, see Reference Literal.

Note: A compound operand is not permitted in a reference literal.

e. Positive A or B

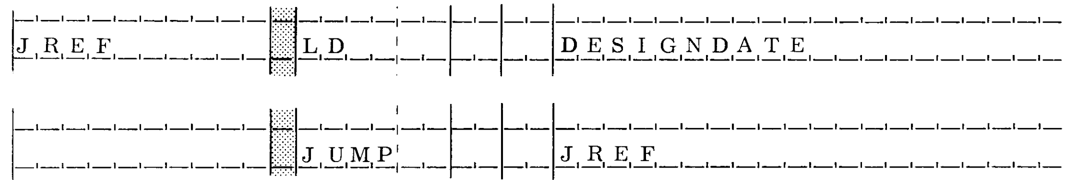
The A (or B) portion of an instruction, which will be added to the contents of an index register to form the full address, must be a positive value. A negative increment is not permitted as part of the coded instruction. When using a compound address, be certain that the result of the operand evaluation does not result in a negative value. For example, using the Data Definition example in Chapter III, the operand: DESIGNDATE-1 would be permitted since the evaluation would result in 003 relative to IR10. However, the operand: DESIGNDATE-4 would result in an error since the result would be negative: -01 relative to IR10.

f. Blank operand

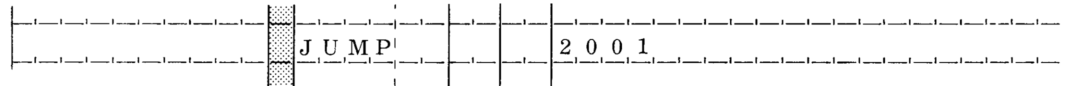
If the Compiler encounters a machine instruction in which a required operand has not been entered, it will construct the absolute coding allowing the correct number of slabs, but will substitute zeros for the missing information and will usually print an error notation on that line of the listing.

2. Descriptive Character of Symbolic Addresses

A name entered in the Reference column for an instruction will be assigned the address of the coding that results from that line on the programming worksheet. When the reference is used as an operand in another instruction, the Compiler will convert the reference into the correct address notation. In the following example:



if the LD instruction were stored in memory at location 02001, the Compiler would treat the JUMP instruction as though it had been written:



In certain cases, additional descriptive information will be associated with the reference. For example, a reference named with the control instruction ALPHA will have associated with it, the address of the Alpha constant, and also the length of the constant.

The reference of a data unit named with the control instruction DATA will have associated with it, the address of the data unit, the length of the data unit, and also the index register to be used if one has been assigned.

When the reference is used as an operand, and the Length and X columns are left blank, the Compiler will supply the correct Length, X and A entries from the descriptive information associated with the reference.

In the case of one or more references used with a compound address, the Compiler will use the descriptive information associated with the first reference.

3. To Modify Descriptive Definitions by Using Length and X Entries

The descriptive definition of a symbolic operand can be modified temporarily without changing the original definition. The memory location, length, and index register associated with a reference can be changed and this change will apply only to the instruction which contains the modifications.

In constructing the absolute coding, the Compiler will use the new values only for this particular instruction. When the reference is again used as the operand of another instruction, and the Length and X columns are left blank, the Compiler will use the descriptive definition as originally assigned.

To modify the memory location referenced by a machine instruction, use a plus or a minus and an integer following the reference. To assign a different length of a different index register, enter the new length or the new index register number in those columns.

V. MACRO INSTRUCTIONS

A. GENERAL

The Compiler can call upon an expandable library of macro subroutines. These are standard subroutines, composed of a number of instructions performed in a given sequence, which are common to many programs and may even be used many times in a single program, varying only the operands. Each subroutine exists in a generalized form as part of a macro generator which is stored in a library (either on CRAM or magnetic tape).

The programmer simply writes a line of coding (referred to as a macro instruction) on the programming worksheet at the point where the subroutine is to be executed in the object program. The macro instruction consists of the name of the subroutine (entered in the Operation column) so that the subroutine can execute its function for this particular program. The Compiler will obtain the desired macro generator from the library, and will transfer control to it.

The generator contains within itself a subroutine in skeleton form. It uses the operands to create the parameters required to perform this subroutine for this particular file or this particular data, and inserts these parameters in the proper positions as part of the machine instructions. The Compiler assigns the appropriate memory locations to this completed routine.

Some subroutines are stored "in-line", that is, they are inserted in their proper place in the sequence of operation. Some subroutines are stored "out-of-line", that is, they are stored away from the routine that refers to them, and are connected to the routine by appropriate jump instructions. If a particular subroutine will be called for at several different points in a program, memory space is conserved by storing the subroutine only once. The subroutine is generated so as to be stored in a reserved area in memory, and appropriate jumps set up from the various points in the main program to the out-of-line coding and back.

Those macro instructions which manipulate files on CRAM or magnetic tape function in conjunction with file tables. These are the tables which were set up in memory by the Compiler from the entries on the File Specification Worksheets. The programmer assigns a name--the file table reference--to each table.

The file tables will contain the parameters pertinent to the various CRAM or magnetic tape files, and will also be used to store data generated during the running of the object program. When the file table reference is entered as an operand of the macro instruction, the Compiler will supply the appropriate addresses of the fields in the file table and insert them in the machine instructions in the generated subroutines.

The number of macro subroutines which the Compiler may utilize is not limited to those initially furnished at the time the Compiler was written or the 315 Computer system installed. Additional macro subroutines may be written as the need arises, and added to the library of macro subroutines.

B. WRITING MACRO INSTRUCTIONS

In general, the rules for entering macro instructions on the programming worksheet are the same as for machine instructions, with the following exceptions:

Operation (columns 19-24)

The Operation column is treated as a single 6-character field. Macro instruction mnemonics are entered left justified starting at column 19.

Length and X (columns 25-26 and columns 27-28)

In most cases, no entry is required in the Length column or in the X column. However, a few of the macro instructions do require an entry in these columns. These are explained in the description of the individual macro instructions.

Operands and Remarks (columns 29-74)

Operands are entered in the same manner as described for machine instructions except that there is no "A,B,Y" as such. When an instruction requires several operands, they are written as a continuous statement separated only by commas. Where a particular operand of an instruction is not required, it is not entered; however, a comma is entered in its place, if it is followed by another operand.

Two consecutive spaces occurring in this column denotes to the Compiler that this is the end of the operands entry and that remarks follow.

Some macro instructions which permit several operands (e.g., FILEC or FILE) may require more space for the operands entry than is contained in one line of the programming worksheet. In this case a long operands entry may be extended to succeeding lines of the programming worksheet. This is done by entering the next line number, leaving the Operation column of the next line blank, and continuing the list of operands in the Operands column starting at column 29. However, each operand (and its comma) must be complete on one line.

For a description of the mnemonics, format, and function of each macro instruction, refer to the Macro Instructions Manual, MD 315-44.

C. SCIENTIFIC SUBROUTINES

A Compiler is available that can call upon an expandable library of Scientific Subroutines. These are standard subroutines that perform a particular function such as: Form the sum of two floating point numbers, evaluate the square root of a fixed point argument, etc. These subroutines are stored in a library on CRAM or on magnetic tape. The standard Compiler does not contain these macros; however, a Compiler that does contain them is available on request.

The programmer simply writes a line of coding on the programming worksheet, at the point where the subroutine is to be executed in the object program. This entry consists of the mnemonic title of the subroutine; e.g., ADD F (which represents Floating Add), or SQRT (which represents Square Root) written in the Op and V columns, and the addresses of the parameters, if any, written in the Operands column.

The Compiler will obtain the desired subroutine from the library and include it in the object program, and will substitute a JUMP to the subroutine, in place of the mnemonic operation code, followed by a series of PAIR's, if appropriate. Each PAIR references a two-slab memory location which contains a parameter address.

For a description of the mnemonics, format, and function of each subroutine, refer to the Scientific Subroutines Manual, MD 315-43.

VI. COMPILER INPUT

The source program consists of the information entered on the Control Worksheet, F-2691, the File Specification Worksheet, F-7304, and the Programming Worksheet, F-2689. This information is keypunched and the resulting punched paper tape or punched card deck becomes the input medium to the Compiler. The Compiler will accept this input and process it as a series of lines (in general, one line equals one instruction to the Compiler).

A. CONTROL WORKSHEET, FORM F-2691

A Control Worksheet, Form F-2691 must be prepared for each program being compiled. For programs being compiled on CRAM, use the side of the form entitled "Control Worksheet, CRAM", and for programs on magnetic tape, use the side entitled "Control Worksheet, Magnetic Tape". The following is a brief description of the columns of the control worksheet. The form itself contains all the information necessary for its completion; however, additional information is given below where appropriate. Differences between the two sides of the form are also noted.

1 4

Page-Line

This sheet should bear the lowest page-line number in the program and it must be the first information input to the Compiler run.

8 16

Program Name

The characters entered here become the name of the object program. (Caution: The rules regarding program names differ slightly from those for other names or references.) The program name may be made up of any of the characters in the 315 Code Chart except the comma. The program name is a ten character field; however, the first 8 characters must contain a unique program name, and the last 2 characters should be used for a program "version number". Spaces are permitted between the last character of the program name and the version number, if the name has less than 8 characters.

Explanation: Assign a program name using columns 8 through 15 on the control sheet. For the initial compilation, the last 2 columns (16 and 17) can be left blank, or the version number "00" can be assigned. For each recompilation, update the version number by "1" (01, 02, 03, etc.). For example:

OBJECT PROGRAM SPECS

8 16

S T O C K 3 0 0 **Program Name**

Initial compilation

19

N E A T

COMPILER INPUT SPECS

23 31

N **Name of Input:**

OBJECT PROGRAM SPECS

First recombilation

8								16		
S	T	O	C	K	3			0	1	

Program Name

19			
N	E	A	T

COMPILER INPUT SPECS

23								31	
S	T	O	C	K	3			0	0

Name of Input:

This will permit identification of versions (recompilations) of the same program; the 8 character name does not change, but the 2 character version number is advanced by the entry on the control worksheet.

At Librarian time, the instructions to the Librarian call for a program by the entire 10 characters. This will insure obtaining the latest version from the program library in the case of several versions of the program appearing on the program CRAM (or tape). The selected program will be included by the Librarian in a new program library.

When the object program is to be run, the System Supervisor will search the program library for the next program to be run, but will search only for an 8 character program name, ignoring the version number. Therefore, it is important that the correct version be on the current program library.

19			
N	E	A	T

The letters "NEAT", representing the control instruction, have been preprinted in these columns and will be punched. NEAT indicates to the Compiler that the information in this line represents the input and output specifications for this Compiler run.

For CRAM:

23 ... 31

Name of Input: If recompiling from an ACRAM, enter Program Name
If not recompiling from an ACRAM, enter **N**

For initial compilation, enter N.

For a recompilation, enter the program name (previous version) if input is from the ACRAM. If input is from punched paper tape or punched cards, enter N.

For magnetic tape:

23 ... 31

Name of Input: If recompiling from an ATAPE, enter Program Name
If not recompiling from an ATAPE, enter **N**

For initial compilation, enter N.

For a recompilation, enter the program name (previous version) if input is from the ATAPE. If input is from punched paper tape or punched cards, enter N.

CONTROL WORKSHEET
CRAM

315 NEAT* COMPILER

Program ______________________ *Prepared by* ______________________

______________________ *Date* ____________ *Page* ___ *of* ___

1 4 [|] Page-Line

75 IDENTIFICATION [ ]

OBJECT PROGRAM SPECS

8 16 [|] **Program Name**

19 [N E A T]

COMPILER INPUT SPECS

23 31 [|] **Name of Input:** If recompiling from an ACRAM, enter Program Name
If not recompiling from an ACRAM, enter **N**

33 [ ] **Indicate type of punched input:** **P** for punched paper tape
C for punched cards
N for no punched input

34 [ ] **Is punched input to be sorted? (Enter Y or N)**

35 [ ] **Are new Page-Line numbers to be assigned? (Y or N)**

COMPILER OUTPUT SPECS

43 [ ] **Type of Executive System which will exercise Primary Control over the Object Program:** **C** for CRAM
T for Magnetic Tape

44 [ ] **Memory Size of Processor on which Object Program is to be run**
(Enter Appropriate Number: 40K = **8**, 30K = **6**, 20K = **4**, 15K = **3**, 10K = **2**, 5K = **1**)

45 [ ] **To which Library should output be added? Enter:** **G** for Macro Generator Library
P for Program Library

46 [ ] **Indicate type of punched output desired:** **P** for punched paper tape
C for punched cards
N for no punched output

47 [ ] **Enter:** **F for Initial Compilation or Full Recompilation**
P for Partial Recompilation (No Macro generation occurs on a Partial)

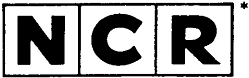
48 [ ] **Should Compiler printout include Cross-Reference listing? (Y or N)**

49 [ ] **Starting Card**
52 [ ] **Ending Card**
} ACRAM Input — Designate location of program to be recompiled
If not recompiling from an ACRAM leave blank

55 [ ] **Starting Card**
58 [ ] **Ending Card**
} ACRAM Output — Designate cards to receive Compiler output

F-2691 ● *TRADEMARK *SERVICE MARK PRINTED IN U. S. A.

Figure 3. Control Worksheet, CRAM



CONTROL WORKSHEET
Magnetic Tape

315 NEAT*
COMPILER

Program ____________ *Prepared by* ____________

____________ *Date* ____________ *Page* ___ *of* ___

1 4 Page-Line

75 IDENTIFICATION

OBJECT PROGRAM SPECS

8 16 **Program Name**

19 N E A T

COMPILER INPUT SPECS

23 31 **Name of Input:** If recompiling from an ATAPE, enter Program Name
If not recompiling from an ATAPE, enter **N**

33 **Indicate type of punched input:**
P for punched paper tape
C for punched cards
N for no punched input

34 **Is punched input to be sorted?** (Enter **Y** or **N**)

35 **Are new Page-Line numbers to be assigned?** (**Y** or **N**)

COMPILER OUTPUT SPECS

43 **Type of Executive System which will exercise Primary Control over the Object Program:**
C for CRAM
T for Magnetic Tape

44 **Memory Size of Processor on which Object Program is to be run**
(Enter Appropriate Number: 40K = **8**, 30K = **6**, 20K = **4**, 15K = **3**, 10K = **2**, 5K = **1**)

45 **To which Library should output be added? Enter:**
G for Macro Generator Library
P for Program Library

46 **Indicate type of punched output desired:**
P for punched paper tape
C for punched cards
N for no punched output

47 Enter:
F for Initial Compilation or Full Recompilation
P for Partial Recompilation (No Macro generation occurs on a Partial)

48 **Should Compiler printout include Cross-Reference listing?** (**Y** or **N**)

F-2691 ● *TRADEMARK *SERVICE MARK PRINTED IN U S A

Figure 4. Control Worksheet, Magnetic Tape

33 ☐ **Indicate type of punched input:** **P** for punched paper tape
C for punched cards
N for no punched input

For initial compilation, enter P or C.

For a recompilation, enter N if the input is from the ACRAM or ATAPE. If input is from punched paper tape or punched cards, enter P or C.

34 ☐ **Is punched input to be sorted? (Enter Y or N)**

If Y, the Compiler will sort the punched input, rearranging the lines into ascending alpha-numeric sequence according to the characters entered in the Page-Line column. The Sort option eliminates the need to rearrange or to externally presort the input.

If N, or if page-line numbers are not assigned, the instructions must be in correct sequence at the time of input to the Compiler run. This is also true for changes, deletions, or additions during a recompilation.

35 ☐ **Are new Page-Line numbers to be assigned? (Y or N)**

If Y, the Compiler will assign new page-line numbers starting with 000000 for the first line and increasing by 10 for each subsequent line.

On the initial compilation, if Sort is also specified (Y in column 34), renumbering will occur after the input is sorted.

For a recompilation, the new input will be sorted, if specified, then merged with the recompilation master (which is assumed to be in correct sequence), and the entire source program will be renumbered. If Sort is not specified, the Compiler will merge the new input, then renumber.

43 ☐ **Type of Executive System which will exercise Primary Control over the Object Program:** **C** for CRAM
T for Magnetic Tape

Enter the appropriate letter to indicate the type of executive routines that will control the object program. For an object program to be run on a mixed system which will require both CRMX-II and STEP, indicate which executive system will exercise primary control (CRMX-II System Supervisor to locate the next program on CRAM, or STEP System Supervisor to locate the next program on magnetic tape).

44 ☐ **Memory Size of Processor on which Object Program is to be run**
(Enter Appropriate Number: 40K = **8**, 30K = **6**, 20K = **4**, 15K = **3**, 10K = **2**, 5K = **1**)

Enter appropriate code number. If this column is left blank, the Compiler will use the memory size of the computer on which this compilation is being run.

45 ☐ **To which Library should output be added? Enter:** **G** for Macro Generator Library
P for Program Library

Enter appropriate letter to indicate which library is to receive output.

46 [ ] **Indicate type of punched output desired:**
- **P** for punched paper tape
- **C** for punched cards
- **N** for no punched output

The object program will be output on CRAM or magnetic tape. If punched paper tape or punched card output is also desired, enter P or C.

47 [ ] **Enter:**
- **F for Initial Compilation or Full Recompilation**
- **P for Partial Recompilation** (No Macro generation occurs on a Partial)

Enter F for the initial compilation.

Enter F for a recompilation when macro instructions are to be added, changed, deleted, etc.

P may be used for a recompilation when macro instructions will not be affected. A partial recompilation takes less time than a full recompilation.

48 [ ] **Should Compiler printout include Cross-Reference listing? (Y or N)**

If Y, the Compiler printout will include a listing of all references, indicating where each is defined and where each is used in the program.

The following columns are for use with CRAM programs only. For compilations to be run using magnetic tape, leave them blank.

For CRAM only:

Column	Field		
49	**Starting Card**	ACRAM Input	Designate location of program to be recompiled If not recompiling from an ACRAM leave blank
52	**Ending Card**		
55	**Starting Card**	ACRAM Output	Designate cards to receive Compiler output
58	**Ending Card**		

75 IDENTIFICATION

Enter identification, if any, as described in Chapter II.

B. CRAM FILE TABLES

In order to use the automatic CRAM handling facilities of CRMX-II and the input-output macro instructions, certain information regarding each CRAM file must be present in memory during the running of the object program. This information is maintained in memory in the form of a table of information, called a file table, for each file.

The file tables contain information to be used for file identification and protection input-output operations, restart procedures, end-of-file procedures, etc., and are constantly being referred to by the CRMX-II subroutines and by the macro instructions. The file tables are also used to store data generated during the running of the object program, e.g., card number in decimal and in binary, address of current record, etc.

File tables are assigned a fixed memory area and are stored in memory in the order generated, normally starting at memory location 00169. (When program overlays are used, the file tables will begin 2 slabs subsequent for each overlay used.) The file tables are produced by the Compiler using information punched from the CRAM File Specification Worksheets.

Figure 6 shows the layout of the file table and indicates the fields in the file table that are affected by corresponding columns in the File Specification Worksheet.

C. CRAM FILE SPECIFICATION WORKSHEET, FORM F-7304

A File Specification Worksheet (CRAM), Form F-7304, must be prepared for each file that is affected by CRMX-II or by any CRAM macro instruction. The form itself contains all the information necessary for its completion; however, additional information is given below where appropriate. For additional detail of the functions of the various executive routines affected, refer to the CRMX-II Manual, MD 315-80.

1 4
Page-Line

Page and Line entries are the same as described in Chapter II.

8
File Table Reference

The characters entered here become the name of the file table for this file and will be used as the operand of instructions to represent the memory location of the file table.

The standard rules for names and references apply to file table references; that is, any letter or numeral but at least one letter and no embedded spaces. File Table Reference entries must be left justified.

When the file table reference is used as the operand of a macro instruction, the Compiler will provide the appropriate addresses for the particular instructions involved.

For addressing purposes, the Compiler treats the 2nd slab of the file table (containing primary CRAM number) as the base address of the file table.

19
F O R M A T C

The letters "FORMAT C", representing the control instruction, have been preprinted in these columns and will be punched. FORMAT C indicates to the Compiler that the information in this line and the next line represents CRAM file table specifications which the Compiler can use to build file tables. This is done by entering the file table reference as an operand of the macro instruction FILEC. FILEC will set up the file table and include it as part of the object program.

Another function of FORMAT C is to identify the CRMX-II options requested for this program. These options are indicated by a Y or N (Yes or No) entered in the appropriate columns of the sheet. The Compiler will store the necessary executive routines on CRAM or in memory.

27
Primary CRAM Number

Enter primary CRAM number.

FILE SPECIFICATION WORKSHEET
CRAM

315 NEAT*
COMPILER

Program __________ *Prepared by* __________

Date ________ *Page* ___ *of* ___

ALL SYMBOLIC REFERENCES MUST BE LEFT-JUSTIFIED AND MUST CONTAIN AT LEAST ONE ALPHABETIC CHARACTER.
ALL ABSOLUTE ADDRESSES OR NUMERIC ENTRIES MUST BE RIGHT-JUSTIFIED AND MUST BE ZERO-FILLED TO THE LEFT.

1 4 **Page-Line** 75 **IDENTIFICATION**

8 **File Table Reference**

19 F O R M A T C

27 **Primary CRAM Number**

29 **Alternate CRAM Number** (If no alternation, enter Primary CRAM Number here also)

31 **Is this a Source File?** (Enter **Y**es or **N**o)

32 **Is Automatic File Setup wanted? (Y or N)**

33 **Is a Rescue Point wanted at the End-of-Section of this File? (Y or N)**

34 **Are "Card Drops" to be time-shared? (Y or N) (N** must be specified if NEXTR or NEXTSP is used, or if this program utilizes only one CRAM unit)

35 **Is this File both a Source and Destination File? (Y or N)** (If Yes, also place a **Y** in Column 31. If No, also place an **N** in Column 36)

36 **Is this File an Overflow File? (Y or N)** (If Yes, Columns 54 thru 71 must be identical to the File Spec Sheet associated with NEXTR)

37

38 **Starting Card Number of File**

41 **Ending Card Number of File**

44 **File Name**

54 **Location of Input/Output Area**

64 **Number of slabs in Input/Output Area** (Maximum–1550 including 4 slabs for the Track Label) (Maximum–1540 including 4 slabs for the Track Label if NEXTR is used)

68 **Record-length is F for Fixed / V for Variable**

69 **Maximum number of slabs in Record**

BRANCH ADDRESSES

1 4 **Page-Line**

29 **Control Mark**
- CSYS!HALT = Halt if CM is other than SK
- Reference = Entry point of Programmer's own-code

39 **Unreadable Block**
- CSYS!HALT2 = Processor halts—Pressing COMPUTE attempts 40 times to read bad block
- CSYS!USE = Use the Block and continue processing
- Reference = Entry point of Programmer's own-code

49 **End-of-Section**
- CSYS!RET = Extremity Routine handles
- Reference = Entry point of Programmer's own-code

59 Associated Index Registers

F-7304 NCR *TRADEMARK *SERVICE MARK PRINTED IN U.S.A.

Figure 5. CRAM File Specification Worksheet

Description of File Table Field	Relative Slab Position	Format		Corresponding Columns on File Specification Worksheet (CRAM)
Number of Slabs in File Table				
Primary Unit Number	0			27 Primary CRAM Number
Card Number in Binary (Select Mode)	1			
Installation Deck Number	2			
Card Number in Decimal (Read or Write Mode)	4			
Track Number	5			
Number of Slabs to Read or Write	6			64 Number of slabs in Input/Output Area
Control Mark Branch Address	8	F-T	F-T	33 Is a Rescue Point wanted at the End-of-Section of this File? (Y or N) 29 Control Mark
Unreadable Block Branch Address	10	F-W	F-W	31 Is this a Source File? (Enter Yes or No) 39 Unreadable Block
End-of-Section Branch Address	12	F-Y	F-Y	32 Is Automatic File Setup wanted? (Y or N) 49 End-of-Section
	14	F-U, F-A	F-U	35 Is this File both a Source and Destination File? (Y or N)
Alternate Unit Number	15			29 Alternate CRAM Number
Card Number in Decimal (Select Mode)	16			
Card Number in Binary (Read or Write Mode)	17			
Starting Card Number of File Section	18			38 Starting Card Number of File
Ending Card Number of File Section	19			41 Ending Card Number of File
End-of-Data Card and Track Number	20			
	22	F-S		
	23	F-AC, F-AD	F-AD	34 Are "Card Drops" to be time-shared?
Date 1 (Used for Label Checking)	24			
Date 2 (Used for Label Checking)	26			
File Name	28			44 File Name
File Section Number	33			

Figure 6. CRAM File Table Showing Fields Affected by Entries on File Specification Worksheet (Sheet 1 of 2)

Description of File Table Field	Relative Slab Position	Format			Corresponding Columns on File Specification Worksheet (CRAM)
Address of Input/Output Area	34	F-AE			54 Location of Input/Output Area
	36	F-V	F-AF	F-V	68 Record-length is F for Fixed, V for Variable
Maximum Number of Slabs in Record	37				69 Maximum number of slabs in Record
Source File - Contents of IR 30; Destination File - Address of Output Area plus Number of Slabs in Output Area	38				
Address of Current Record	40				
	42				

This sentinel follows the last File Table in the program and indicates the end of the File Table area: - 0 0

CRAM FILE TABLE (cont'd)

Flag	Designations if ON
F-A	Dropped card was not verified (Write mode only)
F-S	File Table has been initialized
F-T	Rescue Dump wanted at End-of-Section
F-U	This is a Source-Destination file
F-V	Records are variable in length
F-W	This is a Source File (F-W OFF - Destination File)
F-Y	Automatic File Setup wanted on this file
F-AC	Drop next card allowing one more read or write on present card
F-AD	Do not time share drops
F-AE	Indicates activity on this block
F-AF	In sequential processing - More than one input area In random processing - Overflow has occurred

Figure 6. CRAM File Table Showing Fields Affected by Entries on File Specification Worksheet (Sheet 2 of 2)

29 Alternate CRAM Number (If no alternation, enter Primary CRAM Number here also)

Enter alternate CRAM number.

31 Is this a Source File? (Enter Yes or No)

If this is a source file, enter Y.
If this is a source-destination file, enter Y; also enter Y in column 35.
If this is a destination file, enter N.

32 Is Automatic File Setup wanted? (Y or N)

Y = The System Supervisor will open this file. Automatic file setup will not initialize the associated index registers of this file (columns 59 through 66 of the second "line" of the file specifications).

N = Automatic file setup is not wanted. If N, use the macro instruction OPENC to open this file. OPENC will initialize the associated index registers.

33 Is a Rescue Point wanted at the End-of-Section of this File? (Y or N)

Y = For multi-deck files, the Extremity Routine will establish a rescue dump at the end of each section on the alternate unit according to the indicated memory size: Card 255 in 5 or 10K; Cards 254 and 255 in 15 or 20K; Cards 252, 253, 254 and 255 in 40K.

N = No rescue dump is wanted at the end of each section.

34 Are "Card Drops" to be time-shared? (Y or N) (N must be specified if NEXTR or NEXTSP is used, or if this program utilizes only one CRAM unit)

Y = The next card in sequence is dropped just prior to the last read or write on the present card.

N = Time-sharing is not wanted. Enter N if the macro instruction NEXTR or NEXTSP is used. Enter N if the macro instructions NEXTIN and NEXTOT are used with the same CRAM unit.

35 Is this File both a Source and Destination File? (Y or N) (If Yes, also place a Y in Column 31. If No, also place an N in Column 36)

Y = This is a source-destination file (will write on the same CRAM cards that were read). Y in column 35 will turn on Flag U in the file table, which will permit writing on this file even though it is a source file.

36 Is this File an Overflow File? (Y or N) (If Yes, Columns 54 thru 71 must be identical to the File Spec Sheet associated with NEXTR)

An overflow file is used in connection with a source-destination file being processed randomly. When an input block is processed using the macro instruction NEXTR, it may be expanded so that it exceeds (overflows) the capacity of the track from which it was read. In this case, part of the block is written back on the original track and the remainder is written on the next available card and track of the overflow file.

Y = This is an overflow file. Y in column 36 indicates that this file (consisting of CRAM card numbers specified in columns 38 through 43) will be used to write this overflow. If Y, columns 54 through 71 must be identical to the file specification worksheet associated with NEXTR.

37

This column is not used.

38 **Starting Card Number of File**

If cards are assigned by the programmer, enter the number of the first card assigned to this file. Cards assigned must be consecutive and will include all cards between starting card and ending card (columns 41 through 43). If cards are not assigned, enter N.

41 **Ending Card Number of File**

Enter the number of the last card assigned to this file. If cards are not assigned, enter N.

44 **File Name**

The characters entered here become the name of the file itself. The file name will be stored in the file table and will appear eventually in the file directory. It need not be the same name as the file table reference.

54 **Location of Input/Output Area**

This is the name assigned to the area in memory used by this file and must also be entered in the Reference column of the level 1 entry in the Data Definitions for this file.

64 **Number of slabs in Input/Output Area** (Maximum—1550 including 4 slabs for the Track Label) (Maximum—1540 including 4 slabs for the Track Label if NEXTR is used)

Enter the block size, including track label.

68 **Record-length is F for Fixed / V for Variable**

F = Record-length is fixed.

V = Record-length is variable. When using variable length records, the record-length must be contained in the first slab of the record. This one-slab data unit must be shown in the Data Definitions for this file.

69 Maximum number of slabs in Record

When macro instructions are used in conjunction with file tables, the maximum CRAM record length is restricted to 999 slabs. Do not confuse this with block size (columns 64 through 67).

75 IDENTIFICATION

Enter identification, if any, as described in Chapter II.

BRANCH ADDRESSES

1 4 **Page-Line**

This entry should be the next Page-Line number after the entry at the top of the File Specification Worksheet. This is necessary because the file specifications must be punched on two lines, and these cards must be consecutive.

29 **Control Mark**

CSYS!HALT = Halt if CM is other than SK
Reference = Entry point of Programmer's own-code

If the programmer wishes to provide for a control mark other than TT, CC or ←←, he may enter a reference to his own coding. However, he must provide for a return to the main program.

39 **Unreadable Block**

CSYS!HALT2 = Processor halts—Pressing COMPUTE attempts 40 times to read bad block
CSYS!USE = Use the Block and continue processing
Reference = Entry point of Programmer's own-code

49 **End-of-Section**

CSYS!RET = Extremity Routine handles
Reference = Entry point of Programmer's own-code

If the programmer wishes to resolve the above conditions with his own coding, he may do so by entering the reference to his own subroutine. Otherwise he should specify one of the alternatives listed.

59 Associated Index Registers

Enter the index register(s) that must contain the base address of each record as it is being processed. This is necessary if the macro instructions NEXTIN, NEXTOT, NEXTSP, or NEXTR will be used to advance through the records in the block (usually level 2 in the Data Definitions), or if the macro instruction NEXT will be used to advance through data units within the records (usually level 3 or lower).

D. FILEC (File)

Function: This macro instruction causes the Compiler to build CRAM file tables and to include these file tables as part of the object program.

Format:

REFERENCE		Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
		FILE	C			A , B , C etc

A, B, C, etc.: File table references which the programmer has assigned to name the file tables of the various files. Each name must also be entered as the file table reference (columns 8 through 17) on the CRAM File Specification Worksheet for its associated file.

Action:

The macro will set up file tables in the order listed in the Operands column, the first-named will be set up as File Table 1.

If macro instructions are used to manipulate CRAM decks, there must be a file table in memory for each CRAM file affected. The control instruction FORMAT C identifies CRAM file specifications. The macro instruction FILEC will generate file tables from these specifications. Therefore, a program that requires CRAM file tables must contain the macro instruction FILEC.

A program may contain several FILEC macros. They need not be contiguous. A maximum of nine separate FILEC instructions is permitted.

E. MAGNETIC TAPE FILE TABLES

In order to use the automatic tape handling facilities of STEP and the input-output macro instructions, certain information regarding each magnetic tape file must be present in memory during the running of the object program. This information is maintained in memory in the form of a table of information, called a file table, for each file.

The file tables contain information to be used for file identification and protection, input-output operations, restart procedures, end-of-file procedures, etc., and are constantly being referred to by the STEP subroutines and by the macro instructions. The file tables are also used to store data generated during the running of the object program, e.g., tape block count, address of current record, etc.

File tables are assigned a fixed memory area and are stored in memory in the order generated, starting at memory location 00153. The file tables are produced by the Compiler using information punched from the Magnetic Tape File Specification Worksheet.

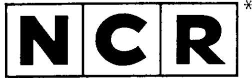
Figure 8 shows the layout of the file table and indicates the fields in the file table that are affected by corresponding columns in the File Specification Worksheet.

F. MAGNETIC TAPE FILE SPECIFICATION WORKSHEET, FORM F-7304

A File Specification Worksheet (Magnetic Tape), Form F-7304, must be prepared for each file that is affected by STEP or by any magnetic tape macro instruction. The form itself contains all the information necessary for its completion; however, additional information is given below where appropriate. For additional details of the functions of the various executive routines affected, refer to the STEP Manual, MD 315-60.

Page and Line entries are the same as described in Chapter II.

The characters entered here become the name of the file table for this file and will be used as the operand of instructions to represent the memory location of the file table. The standard rules for names and references apply to file table references; that is, any letter or numeral but at least one letter and no embedded spaces. File Table Reference entries must be left justified.



FILE SPECIFICATION WORKSHEET
Magnetic Tape

315 NEAT* COMPILER

Program ______________ *Prepared by* ______________

______________ *Date* ______________ *Page* ____ *of* ____

ALL SYMBOLIC REFERENCES MUST BE LEFT-JUSTIFIED AND MUST CONTAIN AT LEAST ONE ALPHABETIC CHARACTER.
ALL ABSOLUTE ADDRESSES OR NUMERIC ENTRIES MUST BE RIGHT-JUSTIFIED AND MUST BE ZERO-FILLED TO THE LEFT.

1 4 **Page-Line** — 75 **IDENTIFICATION**

8 **File Table Reference**

19 F O R M A T T

29 **Primary Handler Number**

30 **Alternate Handler Number.** (If no alternation, enter Primary Handler Number here also)

31 **Is this a Source File?** (Enter **Y**es or **N**o)

32 **Rewind this File with USE-LOCK (Y** or **N)**

33 **Is Rescue Point wanted at the end of every reel of this File, or is the RDUMP Macro used on this File? (Y** or **N)**

34 **Is Automatic File Setup wanted on the FIRST reel of this File? (Y** or **N)**

35 **Insist on WRITE-LOCK on this File (Y** or **N)**

36 **Is Automatic File Setup wanted on the SECOND and subsequent reels of this File, or is the OPEN Macro used on this File? (Y** or **N)**

37 **File Name**

47 **Location of Input/Output Area**

57 **Number of slabs in Input/Output Area**

61 **Record-length is F for Fixed / V for Variable**

62 **Maximum number of slabs in Record**

65 **Associated Index Registers**

BRANCH ADDRESSES

1 4 **Page-Line**

29 **Control Mark**
- **SYS!HALT** = Halt if CM encountered is other than TT, CC or ⋠⋠
- **Reference** = Entry point of Programmer's own-code

39 **Unreadable Block**
- **SYS!HALT** = Processor halts—Pressing COMPUTE drops block, then continues processing
- **SYS!HALT2** = Processor halts—Pressing COMPUTE attempts 8 times to read bad block
- **SYS!DROP** = Drop the Block and continue processing
- **SYS!USE** = Use the Block and continue processing
- **Reference** = Entry point of Programmer's own-code

49 **End-of-Tape**
- **SYS!HALT** = Processor halts
- **SYS!RET** = Extremity Routine handles
- **Reference** = Entry point of Programmer's own-code

F-7304 • *TRADEMARK *SERVICE MARK PRINTED IN U.S.A.

Figure 7. Magnetic Tape File Specification Worksheet

When the file table reference is used as the operand of a macro instruction, the Compiler will provide the appropriate addresses for the particular instructions involved.

For addressing purposes, the Compiler treats the 2nd slab of the file table (containing primary handler number) as the base address of the file table.

19
FORMAT T

The letters "FORMAT T", representing the control instruction, have been preprinted in these columns and will be punched. FORMAT T indicates to the Compiler that the information in this line and the next line represents magnetic tape file table specifications which the Compiler can use to build file tables. This is done by entering the file table reference as an operand of the macro instruction FILE. FILE will set up the file table and include it as part of the object program.

Another function of FORMAT T is to identify the STEP options requested for this program. These options are indicated by a Y or N (Yes or No) entered in the appropriate columns of the sheet. The Compiler will store the necessary executive routines on tape or in memory.

29
Primary Handler Number

Enter primary handler number.

30
Alternate Handler Number. (If no alternation, enter Primary Handler Number here also)

Enter alternate handler number.

31
Is this a Source File? (Enter Yes or No)

If this is a source file, enter Y.
If this is a destination file, enter N.

32
Is Automatic File Setup wanted? (Y or N)

Y = This reel will be rewound with no reading or writing permitted.

N = Reading or writing will be permitted on this reel.

33
Is Rescue Point wanted at the end of every reel of this File, or is the RDUMP Macro used on this File? (Y or N)

Y = The Extremity Routine will establish a rescue dump at the end of each reel in the file.
Also, enter Y if the RDUMP macro is used on this file.

N = A rescue dump is not wanted at the end of each reel; also, the RDUMP macro is not used on this file.

Description of File Table Field	Relative Slab Position	Format		Corresponding Columns on File Specification Worksheet (Magnetic Tape)
Number of Slabs in File Table				
Primary Handler Number	0	Space		29 Primary Handler Number
Number of Slabs to Read or Write	1			57 Number of slabs in Input/Output Area
Tape Block Count	3			
Skip Record Count (Also included in Tape Block Count)	5			
End-of-Tape Branch Address	6	F-S	F-S	32 Rewind this File with USE-LOCK (Y or N); 49 End-of-Tape; 33 Is Rescue Point wanted at the end of every reel of this File, or is the RDUMP Macro used on this File? (Y or N)
Control Mark Branch Address	8	F-T	F-T	29 Control Mark
Unreadable Block Branch Address	10	F-W	F-W	31 Is this a Source File? (Enter Yes or No); 35 Insist on WRITE-LOCK on this File (Y or N); 39 Unreadable Block
Alternate Handler Number	12	F-Y	F-Y	34 Is Automatic File Setup wanted on the FIRST reel of this File? (Y or N); 30 Alternate Handler Number.
Installation Tape Number	13			
	15	F-U blank		
Date 1 (Used for Label Checking)	16			
Date 2 (Used for Label Checking)	18			
File Name	20			37 File Name
Reel Count	25	Space		

Figure 8. Magnetic Tape File Table Showing Fields Affected by Entries on File Specification Worksheet (Sheet 1 of 2)

Description of File Table Field	Relative Slab Position	Format		Corresponding Columns on File Specification Worksheet (Magnetic Tape)
Address of Input/Output Area	27			47 Location of Input/Output Area
Maximum Number of Slabs in Record	29	F-V	F-V	61 Record-length is F for Fixed, V for Variable; 62 Maximum number of slabs in Record; 57 Number of slabs in Input/Output Area
Source File - Contents of IR 30 Destination File - Address of Output Area plus Number of Slabs in Output Area	31			
Address of Current Record	33			
	35			
This sentinel follows the last File Table in the program and indicates the end of the File Table area		- 0 0		

MAGNETIC TAPE FILE TABLE (cont'd)

Flag	Designations if ON
F-S	Rewind reel with USE-LOCK
F-T	Rescue Dump wanted at End-of-Tape
F-U	Restrict acceptable date of all subsequent reels of this file to the "Date Written" in the Tape Label of the first reel -- Source Files only
F-V	Records are variable in length
F-W	This is a Source File (If flag has a value of ⌀ , insist on WRITE-LOCK) F-W OFF - Destination File
F-Y	Automatic File Setup wanted on this file

Figure 8. Magnetic Tape File Table Showing Fields Affected by Entries on File Specification Worksheet (Sheet 2 of 2)

34 ☐ **Is Automatic File Setup wanted on the FIRST reel of this File? (Y or N)**

Y = The System Supervisor will open the first reel of this file. Automatic file setup wiil not initialize the associated index registers of this file (columns 65 through 72 of the file specifications).

N = Automatic file setup is not wanted on the first reel of this file. If N, use the macro instruction OPEN to open this file. OPEN will not initialize the associated index registers.

35 ☐ **Insist on WRITE-LOCK on this File (Y or N)**

Y = This is a source file on which no writing is to be done.

N = Writing is permitted on this file.

36 ☐ **Is Automatic File Setup wanted on the SECOND and subsequent reels of this File, or is the OPEN Macro used on this File? (Y or N)**

Enter Y if this is potentially a multi-reel file. The System Supervisor will handle the first reel of this file, and Y in column 36 will insure that the Extremity Routine will be present to handle the second and subsequent reels.

37 **File Name**

The characters entered here become the name of the file itself. The file name will be stored in the file table and will appear eventually in the magnetic tape label. It need not be the same name as the file table reference.

47 **Location of Input/Output Area**

This is the name assigned to the area in memory used by this file and must also be entered in the Reference column of the level 1 entry in the Data Definitions for this file.

57 **Number of slabs in Input/Output Area**

Enter the block size; maximum is 7999 slabs.

61 ☐ **Record-length is F for Fixed / V for Variable**

F = Record-length is fixed.

V = Record-length is variable. When using variable length records, the record-length must be contained in the first slab of the record. This one-slab data unit must be shown in the Data Definitions for this file.

62

Maximum number of slabs in Record

When macro instructions are used in conjunction with file tables, the maximum magnetic tape record length is restricted to 999 slabs. Do not confuse this with block size (columns 57 through 60).

65

Associated Index Registers

Enter the index register(s) that must contain the base address of each record as it is being processed. This is necessary if the macro instructions NEXTI or NEXTO will be used to advance through the records in the block (usually level 2 in the Data Definitions), or if the macro instruction NEXT will be used to advance through data units within the records (usually level 3 or lower).

75 IDENTIFICATION

Enter identification, if any, as described in Chapter II.

BRANCH ADDRESSES

1 4

Page-Line

This entry should be the next Page-Line number after the entry at the top of the File Specification Worksheet. This is necessary because the file specifications must be punched on two lines, and these cards must be consecutive.

29

Control Mark

- SYS!HALT = Halt if CM encountered is other than TT, CC or ←←
- Reference = Entry point of Programmer's own-code

If the programmer wishes to provide for a control mark other than TT, CC or ←←, he may enter a reference to his own coding. However, he must provide for a return to the main program.

39

Unreadable Block

- SYS!HALT = Processor halts—Pressing COMPUTE drops block, then continues processing
- SYS!HALT2 = Processor halts—Pressing COMPUTE attempts 8 times to read bad block
- SYS!DROP = Drop the Block and continue processing
- SYS!USE = Use the Block and continue processing
- Reference = Entry point of Programmer's own-code

49

End-of-Tape

- SYS!HALT = Processor halts
- SYS!RET = Extremity Routine handles
- Reference = Entry point of Programmer's own-code

If the programmer wishes to resolve the above conditions with his own coding, he may do so by entering the reference to his own subroutine. Otherwise he should specify one of the alternatives listed.

G. FILE (File)

Function: This macro instruction causes the Compiler to build magnetic tape file tables and to include these file tables as part of the object program.

Format:

REFERENCE		Op	V	LENGTH / LEVEL	X	INSTRUCTIONS: OPERANDS / DATA DEFINITIONS: LENGTH, TYPE
8 9 10 11 12 13 14 15 16 17	18	19 20 21 22	23 24	25 26	27 28	29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
		FILE				A, B, C etc

A, B, C, etc.: File table references which the programmer has assigned to name the file tables of the various files. Each name must also be entered as the file table reference (columns 8 through 17) on the Magnetic Tape File Specification Worksheet for its associated file.

Action:

The macro will set up file tables in the order listed in the Operands column, the first-named will be set up as File Table 1.

If macro instructions are used to manipulate magnetic tapes, there must be a file table in memory for each magnetic tape file affected. The control instruction FORMAT T identifies magnetic tape file specifications. The macro instruction FILE will generate file tables from these specifications. Therefore, a program that requires magnetic tape file tables must contain the macro instruction FILE.

A program may contain several file macros. They need not be contiguous.

H. CONVENTIONS

1. Sequence of Input

a. NEAT

The control instruction NEAT, which identifies the Compiler input and output specifications, must be the first instruction physically input to the Compiler run.

b. END

The control instruction END, which marks the end of the source program, must be the last instruction physically input to the Compiler run, and if Sort is requested, must also bear the highest page-line number.

c. ENDMOD

The control instruction ENDMOD, which marks the end of the correction deck in a recompilation, must be the last instruction physically input to the recompilation run.

d. FINISHC or FINISH

The macro instruction FINISHC (for CRAM) or FINISH (for magnetic tape) must follow the last instruction to be executed in the object program. The Compiler will substitute the necessary instructions to locate and read in the next program to be run and perform the necessary pre-run setup.

2. Other Requirements

a. FORMAT C and FILEC

If macro instructions are to be used to manipulate CRAM files, the source program must contain CRAM file specifications (identified by the control instruction FORMAT C). The source program must also contain the macro instruction FILEC to direct the Compiler to build CRAM file tables.

b. FORMAT T and FILE

If macro instructions are to be used to manipulate magnetic tape files, the source

program must contain magnetic tape file specifications (identified by the control instruction FORMAT T). The source program must also contain the macro instruction FILE to direct the Compiler to build magnetic tape file tables.

I. CHANGES

The Compiler will treat lines with identical page-line numbers according to the following chart. These rules apply to recompilations as well as to the initial compilation.

SORT or NO SORT	PUNCHED INPUT	LINES ARE	COMPILER WILL USE
sort	paper tape	adjacent	last line input
		not adjacent	last line input
	cards	adjacent	last line input
		not adjacent	last line input
no sort	paper tape	adjacent	last line input
		not adjacent	all lines input
	cards	adjacent	all lines input
		not adjacent	all lines input

If Sort is specified, changes and corrections can be made by entering the correct line of coding with the page-line number of the line to be changed. The Compiler will substitute the second line (the correct line) for the original line during the sort.

If Sort is not specified, the rules vary depending on the input and whether or not the lines are adjacent.

J. DELETIONS

For the initial compilation (if Sort is specified), to delete a line, simply enter the page-line number of the line to be dropped, and use the control instruction XXX.

For recompilations (whether Sort is specified or not), to delete one or more lines, enter the page-line number(s) of the line(s) to be dropped, and use the control instruction OMIT. (OMIT may be used to delete a single line in the initial compilation.)

Note: The control instructions NEAT, END, ENDMOD, FORMAT C, FORMAT T, XXX, and OMIT are described elsewhere in this manual. The macro instructions FINISHC, FINISH, FILEC, and FILE are detailed in the Macro Instruction Manual.

VII. COMPILER OUTPUT

A. CRAM OR MAGNETIC TAPE

1. Object Program

The primary output of the Compiler is the object program. This program is generated as a series of blocks written on CRAM (referred to as ACRAM), or on magnetic tape (referred to as ATAPE).

The object program can then be placed from the ACRAM or the ATAPE into a library of programs by the CRAM Librarian, or by the Magnetic Tape Librarian.

2. Recompilation Master

The Compiler will provide a recompilation master on the ACRAM or ATAPE. This contains the source program as written by the programmer. It also contains the instructions and out-of-line coding generated by the Compiler, e.g., control instructions to set up file tables and out-of-line constants, machine instructions and subroutines called for by macro instructions, etc. During a partial recompilation, the Compiler will read the source program from the recompilation master, and the programmer may use punched cards or tape to make corrections to the source program which he has written, provided these corrections do not affect the macro instructions used in the program. If the macro instructions are affected, the programmer must make a full compilation using the punched medium input or the recompilation master.

B. OPTIONAL PUNCHED OUTPUT

1. Punched Paper Tape

If requested by the programmer (P in column 46 of the control sheet), the Compiler will provide additional output in the form of punched paper tape containing the object program.

2. Punched Cards

If requested by the programmer (C in column 46 of the control sheet), the Compiler will provide additional output in the form of punched cards containing the object program.

C. PRINTOUT

An important function of the Compiler is to produce a printout which will include the entire source program and the object program showing the absolute addresses allocated for each instruction, constant, data unit, etc., and the contents of these memory locations.

The printout is a valuable aid to the programmer. It provides a picture of internal memory and a complete record of his program. It enables him to locate and correct errors in his program and to make whatever additions, deletions, or substitutions may be necessary.

The following is a brief description of the sections usually included in a Compiler printout.

The pages of the printout are numbered starting with Page 1. Each page has an identical header format, showing page numbers, program name, date compiled, and the program version number. Each page also has column headings which identify the information contained in that column. These headings vary, depending upon the section of the printout.

In general, an instruction line written on the programming worksheets will be shown as a line in the printout. These lines bear the page-line number assigned by the programmer. Instructions that are generated by the Compiler are assigned a unique number called a sequence number. These can be identified by an asterisk, space, and a four-digit sequence number printed in the Page-Line column (for example: * 1234). For literal constants generated out-of-line by the Compiler, the Page-Line column will be blank.

Errors in the program that have been detected by the Compiler may be shown in several ways:

as part of the listing of errors,

as a remark line printed in its appropriate place relative to the error,

as a symbol printed at the left on the line in which the error occurs.

Error Notations:

Symbol	Signifies
*	Illegal Operation code
* L	Error in the L field
* X	Error in the X field
* A	Error in the A operand
* B	Error in the B operand
* Y	Error in the Y operand
* S	A or B operand undefined
* R	Range of operand illegal

A printout may contain all or some of the following listings in the order indicated below:

1. Errors

   In alphanumeric order by operation code.

   Shows type of error, page-line number, operation code.

   Some examples of these errors are: illegal codes punched in paper tape or cards, illegal literals, illegal operands, and illegal operation codes.

2. HALT instructions

   In alphanumeric order by page-line or sequence number.

   Shows page-line or sequence number and complete instruction (reference, operation code, etc.).

3. TEST:SW instructions

   In alphanumeric order by page-line or sequence number.

   Shows page-line or sequence number and complete instruction.

4. All Symbolic References (Cross-Reference Listing)

   In alphanumeric order by reference name.

   Shows page-line or sequence number where the reference was defined, memory loca-

tion corresponding to the reference, points in the program (page-line or sequence numbers) which use the reference as an operand.

Undefined references and duplicate references are also shown.

Note: Chapter II states that a reference name may be made up of any of the letters of the alphabet (A to Z) and any of the decimal numbers (0 to 9). However, note the presence of the following symbols: exclamation point, semi-colon, quote (! ; ") in some of the references in the printout. The Compiler will accept them as part of a reference; however, they should not be used by customer programmers. They are reserved for use with macro subroutines, CRMX-II, STEP, Librarian, Sort Generators, etc., in order that these system references will remain unique.

5. File Tables

In numeric order by file table number, starting with File Table 1. File tables for object programs to be run using CRAM will start at memory location 00169, and at location 00153 for magnetic tape. If the object program makes use of overlays, the starting location for CRAM file tables will be two slabs greater than 00169 for each overlay used.

Indicates options requested on the file specification sheet.

Shows memory location of each data unit in the file table, and actual contents of that location in Digit form. On the same line, shows the sequence number and the complete control instruction that set up the data unit.

A one slab sentinel follows the last file table in the program and indicates the end of the file table area. This sentinel is represented as -00.

Program Safe Area begins here.

6. Literal constants generated out-of-line by the Compiler.

In alphanumeric order by operation code, L, and operand.

Shows the memory location of the constant, the contents of that location in Digit form, and the control instruction that set up the constant.

7. Out-of-Line Coding

These are closed subroutines which are furnished due to the programmer's use of various macro instructions.

In alphanumeric order by macro name.

Shows memory location of the instruction, contents of that location (actual instruction in absolute format), evaluated instruction, sequence number and complete machine instruction in mnemonic form, including remarks.

The column heading: EV.A/B indicates a six-character column representing the evaluated A or B operands, and contains:

A address plus contents of the X register,
or B address plus contents of the Y register.

The column heading: CD XY A/B represents the evaluated absolute format of the

instruction, and contains:

C = C of the instruction, or G if double stage
D = F of the instruction, or Q if double stage

Note: Single stage -- D = Length or F, not xF
Double stage -- D = F or Q (variation of C)

The EQUATEs generated at this point by the FINISHC macro, or the FINISH macro. Shows EV.A/B, sequence number, and complete EQUATE instruction.

These EQUATEs relate the system references to fixed memory locations, such as memory flags in the Universal Safe Area, addresses in the Kernel, etc. They occur where the FINISHC macro, or the FINISH macro, would appear alphanumerically among the listing of out-of-line coding. The balance of the out-of-line macro subroutines follows the EQUATEs.

From this point on, the program is printed in the order that the lines are input to the Compiler (physical order or alphanumeric sequence by page-line number if sorted).

8. FORMAT C lines, or FORMAT T lines

   Shows the contents of each file specification worksheet, printed on two lines.

9. FILEC lines, or FILE lines

   Shows the page-line number and the complete instruction as written on the programming worksheets.

10. DATA, INDEX, and REDFN lines

    Shows the complete Data Definition lines as written on the programming worksheets.

11. In-line Constants

    Shows the memory location of the constant, contents of that location in Digit form, the page-line number and complete control instruction (ALPHA, DIGIT, PAIR, etc.) used by the programmer to set up the constant.

12. The subroutines which contain the instructions to be executed during the running of the object program may be written to occupy a single portion of the printout; or they may be interspersed with control instructions that set up constants and data definitions.

    Each instruction line shows the memory location of the instruction, the contents of that location (actual instruction in absolute format), the evaluated instruction, the page-line number, and the complete machine instruction in mnemonic form. In-line (open) subroutines furnished for macro instructions are also included wherever the macro instruction appears.

13. END line

    This is the last line printed. It shows the actual Program Starting Address and the complete END control instruction as written on the programming worksheet (page-line number, operation code END, Program Starting Address).

VIII. KEYPUNCHING PROCEDURES

A. GENERAL

The Compiler forms an 80-character image in memory for each line of the source program; each line corresponds to one line punched into paper tape or to one punched card.

In punching from the file specification worksheets and from the programming worksheets, it is not necessary to hit the space bar for each blank character position on the line. Instead, stops corresponding to the columns of the line can be set up, and in many cases it is possible to skip from one column to the next. These stops correspond to the columns of the programming worksheet:

Position	Column
4	Line
8	Reference
19	Operation
25	Length (or Level)
27	X
29	Operands (or Length, Type)
49	Optional left margin for Remarks

Keypunching procedures for paper tape and for cards differ slightly. Aside from the differences caused by the characteristics of the punching equipment, the only variations are in punching the Page and Identification columns.

Caution: The program must be punched exactly as it appears on the sheets. A spelling or a punching error usually causes errors in the compiled program. Also, in punching the Operands entries, be careful not to punch spaces between the operands and the commas that separate the operands.

B. PAPER TAPE

1. General

Set the left-hand margin of the tape-punching typewriter at a position which will accommodate at least 75 characters to a line. Set the tab stops at the positions shown above relative to the left margin. If the typewriter being used has a right-hand margin stop, or automatic carriage-return, set it at position 75.

Each time the tab key is pressed, a specific configuration is punched into the paper tape. The Compiler recognizes this configuration (where it is permitted) and automatically fills out the current column with spaces. When not using the tab key to tab across blank columns, type a space wherever a blank appears or wherever the programmer has written a space symbol (ø).

Each line of typing, corresponding to a line of coding, is terminated by a carriage-return immediately after the last character of the line. There need not be a tab before this carriage-return. The Compiler recognizes the carriage-return punched into the paper tape, and fills out its own image of the line with spaces.

Where the page number is the same as that on the previous line, it need be punched only for the first line on each sheet; for successive lines, merely tab across the Page column to punch the line number. Whenever the Compiler finds a blank entry for Page, it automatically fills in the page number of the preceding line.

Identification is never required. If there is an entry in the Identification column, it should be ignored.

At the beginning and end of each paper tape program, punch at least two feet of "run-in" code by holding down the tape feed key. (In NCR General-purpose Code, this is a punch in channels 1, 2, 3, 4, 5, and 6.) This serves as a leader and a trailer. Several inches of tape feed should also be punched between sheets.

2. Control Worksheet, Form F-2691

Punch this sheet as a single continuous line of characters plus carriage-return. Do not use the tab in this line. Punch spaces wherever necessary to fill out blanks.

3. File Specifications Worksheet, Form F-7304

This sheet contains two lines. Punch the first line as a continuous line of characters. Do not use the tab in this line. Punch spaces wherever necessary to fill out blanks. End the line with a carriage-return.

In punching the second line, tab to position 4 and type the line number in positions, 4, 5, 6. Then tab five times, or space, to position 29, and punch the remainder of the sheet as a continuous line of characters using spaces wherever necessary to fill out blanks. End the line with a carriage-return.

4. Programming Worksheet, Form F-2689

This sheet contains a maximum of 32 lines. Wherever the Page column has been left blank or where the page number is the same as on the previous line, tab across it to position 4 and punch the line number.

Tab to fill out any column, or to skip over a blank column. In some cases it may be just as convenient to use spaces to get to the next column.

Carriage-return at the end of each line, after the last character punched.

a) Punching the Reference column

Note that positions 7 and 18 contain spaces. After punching the line number, punch a space in position 7 to get to the Reference column, then punch the reference. If there is no Reference entry, punch a space in position 7, then tab once to the Operation column.

If there is a Reference entry, punch the reference and tab once to the Operation column. If spaces are used to fill out the Reference column, or if the reference contains a full 10 characters, remember the space at position 18. Punch the Operation entry starting at position 19.

b) Punching the remarks

The remarks may begin anywhere on a line, after position 29, provided there are at least two spaces between the last operand (or the end of the Length, Type entry) and the remarks. In order to obtain an easily-read printout, many programmers prefer to establish a specific left margin for remarks and will start their remarks at position 49.

If the remarks start at position 49, tab to this position after punching the Operands, and then punch the remarks. Sometimes the Operands will extend past position 46 and make it impossible to start the remarks at position 49. (There must be at least two spaces punched between the last operand and the remarks.) In that case, punch two spaces after the Operands and then start the remarks.

If the left margin of the remarks is further to the right, tab to position 49 and then punch the proper number of additional spaces.

If the left margin is desired to the left of position 49, space across to the specified position, and do not use the tab in this part of the line.

The NCR Programming Worksheet has been designed so that the exact number of characters may be entered per line. However, if a less precise programming sheet is being used, the remarks may at times contain enough characters to extend past position 74. In this case:

- Terminate the remarks at position 74, or after the last complete word before that. Carriage-return.
- Tab to position 4. Punch a new line number, intermediate between the previous number and the next one.
- Tab to position 8. Punch an asterisk (*).
- Tab four times to position 29 (or five times to position 49 if desired) and continue the remarks.
- Carriage-return after the last character of the remarks. If the remarks have also overflowed this line, repeat the procedure as often as necessary, using a new intermediate line number each time.

NOTE: An asterisk in position 8 indicates that the entire line is a remark. In this case, punch an asterisk in position 8, and punch the line exactly as it is written.

c) Punching the operands

There will occasionally be cases in which operands would extend past position 74. The procedure for extending operands is the same as for remarks, except:

- Each operand (and its comma) must be complete on one line.
- On successive lines, leave the Reference column blank; leave the Operation column blank.
- Begin the next operand at position 29.

5. Correcting Punching Errors

If the error is detected before starting the next line, carriage-return, tab to position 4, and repunch the entire line correctly. Whenever the Compiler finds two or more consecutive lines with the same page-line number, it discards all but the last line.

If the error is detected after starting to punch the next line, make a note of the page-line number, and notify the programmer. He will decide whether to make the correction during the initial compilation, during a recompilation, or during the Librarian run.

6. End of Paper Tape

If the end of a reel of paper tape occurs before all the program has been punched, punch the control character "Ʒ" (a single punch in paper tape channel 3), and continue punching the program in a new tape.

Be sure to leave at least two feet of run-in code at the end of the reel, and at the beginning of the next reel.

C. PUNCHED CARDS

1. General

Set up the program card with skip-stops corresponding to the columns on the programming worksheets (4, 8, 19, 25, 27, 29, 49 and 75).

Set up the program card to duplicate identification (columns 75 through 80).

The 80 columns on a card correspond exactly to the 80 positions on the program sheets. Punch the entries exactly as written, leaving blanks where the column on the sheet is blank or where a space symbol (⌀) has ben entered. Use the skip-stops to skip across all blank columns. The Compiler interprets blank card columns as spaces.

The Page column must be punched on every card. If the programmer has left the column blank, this signifies that the page number for this card is the same as for the previous card.

If an identification has been entered, this must be punched on every card. There is usually only one identification assigned for the entire program.

2. Control Worksheet, Form F-2691

   Punch the entries on this sheet (including identification) into one card.

3. File Specification Worksheet, Form F-7304

   Punch two cards for each file specification worksheet. Use the dup key to duplicate the page number when it is the same as that on the previous card. Use the skip key to skip across blank columns.

4. Programming Worksheet, Form F-2689

   This sheet contains a maximum of 32 lines. In general, one line equals one card. Use the dup key to duplicate the page number when it is the same as that on the previous card. Use the skip key to skip across blank columns.

   a) Keypunching the remarks

      The remarks may begin anywhere on a line, after column 29, provided there are at least two spaces between the last operand (or the end of the Length, Type entry) and the remarks. In order to obtain an easily-read printout, many programmers prefer to establish a specific left margin for remarks and start their remarks at column 49.

      If the remarks start at column 49, skip to this column after punching the operands, and then punch the remarks. Sometimes the operands will extend past column 46 and make it impossible to start the remarks at column 49. (There must be at least two spaces between the last operand and the remarks.) In that case, leave two blanks after the operands and then start the remarks.

      If the left margin of the remarks is further to the right, skip to column 49 and then leave the proper number of additional blanks.

      If the left margin is to the left of column 49, use the space bar to move over to the specified column, and do not use the skip key in this part of the card.

      The NCR Programming Worksheet has been designed so that the exact number of characters may be entered per line. However, if a less precise programming sheet is being used, the remarks may at times contain enough characters to extend past column 74. In this case:

      - Terminate the remarks at column 74, or after the last complete word before that. Duplicate the identification, then release the card and feed the next one.
      - Duplicate the page number. Punch a new line number, in column 4, 5, 6, intermediate between the previous number and the next one.
      - Skip to column 8. Punch an asterisk (*).
      - Skip four times to column 29 (or five times to column 49 if desired) and continue the remarks.
      - Duplicate the identification, then release the card and feed the next one. If

the remarks have also overflowed this card, repeat the procedure as often as necessary using a new intermediate line number each time.

NOTE: An asterisk in column 8 indicates that the entire line is a remark. In this case, punch an asterisk in column 8, and punch the line exactly as it is written.

b) Keypunching the operands

There will occasionally be cases in which operands would extend past column 74. The procedure for extending operands is the same as for remarks, except:

- Each operand (and its comma) must be complete on one card.
- On successive cards, leave the Reference column blank; leave the Operation column blank.
- Begin the next operand at column 29.

5. Correcting Keypunching Errors

The easiest method to correct a keypunching error is to repunch the entire card and to substitute this correct card for the incorrect one.

6. Keypunching Non-IBM Characters

The NCR 315, with a full character set of 64 configurations, contains 16 characters which are not available on standard IBM 026 keypunches.

These configurations may easily be obtained by overpunching according to the following table. For convenience, the semi-colon is also included in the table, since it corresponds to the IBM character □.

315 CHARACTER		PUNCH	HOLE CODE
exclamation point	!	7 8	7, 8
semi-colon	;	□	12, 4, 8
quotation mark	"	5 8	5, 8
question mark	?	1 2 7	1, 2, 7
colon	:	5 7	5, 7
left arrow	←	2 4	2, 4
up arrow	↑	G H	12, 7, 8
plus sign	+	- 0	11, 0
equal sign	=	V Y	0, 5, 8
left parenthesis	(	3 4 5	3, 4, 5
right parenthesis	)	H Y	12, 0, 8
less than	<	C Z	12, 0, 3, 9
greater than	>	& 0	12, 0
apostrophe	'	P Q	11, 7, 8
left bracket	[	N Q	11, 5, 8
right bracket	]	X Y	0, 7, 8
reverse slant	\	E H	12, 5, 8

TABLE II. NCR 315 CHARACTER PUNCH CONFIGURATIONS

IX. INDEX

Actual Machine Code 1
Addressing of Data 23, 48
ALPHA 31
Alpha Literal 37
Assembly Function 2
Asterisk Address 49
Automatic Programming 1

BASE 44
Blank Operand 50

Changes 75
Compound Address 48
Concept of Levels 17
Control Instructions 3, 11, 31
Control of Location Counter 39
Control Worksheet 54, 56, 57, 81, 83
Conventions 74
CRAM File Specifications 47, 60
CRAM File Tables 59
CRAM Track Labels 29

DATA 25
Data Definition 16, 18
Definition of Constants 31
Deletions 75
DIGIT 32
Digit Literal 37
Double Stage Instructions 14
Dynamic Addresses 24, 49, 50

END 48, 74
ENDMOD 48, 74
EQUATE 43
Executive Routines 5

FILE 73, 74
FILEC 66, 74
File Specification
  Worksheet 60, 61, 62, 67, 68, 70
File Tables 5, 52, 59, 67
FINISHC or FINISH 74
FORMATC or FORMATT 74

Identification 23
INDEX 26
Indexing 24

JVAL 45

Keypunching Procedures 80

Length 11, 20, 51, 53
Level 20
Librarian 5
LIST 46
Literal Operand 13, 36
LITORG 42

Machine Instructions 11, 13
Macro Instructions 4, 11, 52
Magnetic Tape File Specifications . . 47, 67
Magnetic Tape File Tables 67

Names and References . . 10, 19, 43, 50, 81
NEAT 47, 55, 74
NUMBER 33
Numeric Literal 38

Object Program 2, 76
OMIT 46, 75
Operands 12, 13, 36, 50, 82, 83
Operation (OpV) 10, 19, 52
Optional Punched Output 76
Organization of Data 16
ORIGIN 40
OVRLAY 41

PAGE 45
Page and Line 8, 19, 54, 60, 67
PAIR 34
Printout 3, 45, 76
Program Name 54
Programming Worksheet . . 8, 9, 19, 81, 83

Recompilation 46, 48, 74
Recompilation Master 76
REDFN 27
Reference Literal 39
Reference Literal Address 49
Relative Nature of 315 Addressing 23
Remarks 3, 12, 13, 20, 23, 81, 83

SAVE 42
Scientific Subroutines 5, 53

INDEX (Cont.)

SGL 36
Simple Address 48
Single Stage Instructions 14
SLAB 36
Static Addresses 23
Symbolic Coding 2

To Delete a Line During
  Initial Compilation 46, 75
To Delete Lines During
  a Recompilation 46, 75

UNLIST 46

Variable Length Records
  CRAM 30
  Magnetic Tape 30

X 12, 20, 51
XXX 46, 75

An Educational Publication
Marketing Services Department
THE NATIONAL CASH REGISTER COMPANY • DAYTON, OHIO 45409

SP-1141 [114100] 0267