

THAT

A LANGUAGE MANUAL

COMPUTATION CENTER
PROGRAMMING STAFF

*David Chou
162 Arla Drive
Pittsburgh 20, Penna.*



David Chou

**David Chou
162 Arla Drive
Pittsburgh 20, Penna.**

Covers for this
manual will be
available (after 2/16/65) without charge
from the USER consultant
at Scaife Hall 316

CHAPTER 1 - ELEMENTS OF 'THAT'

1.0 INTRODUCTION

'THAT' is a symbolic assembly language designed for writing programs in the machine language of the Central Processor of the CDC G-21 computer system at the Carnegie Institute of Technology Computation Center. This manual will describe the 'THAT' language and the associated Assembly Program, which were developed by the staff of the Computation Center. The reader may refer to the G-20 machine language reference manual ('Central Processor/Machine Language Manual', CDC G-20 Publication No. 611) for information on the logical organization, word formats, arithmetic rules, addressing scheme, and operations of the Central Processor. SECTION 2 of the User's Manual describes the hardware modifications which have been made to the Carnegie Tech system, converting it from a G-20 into a G-21.

The 'THAT' Assembly Program (or "Assembler") accepts a source program containing code in the 'THAT' language, and translates ("assembles") it into absolute binary machine language in core memory. This translation process is generally one-for-one; thus, each 'THAT' statement, occupying a separate line or "card image" of the source program, is translated generally into a single absolute binary instruction or data word. For this reason, 'THAT' is called an "assembly" language.

The 'THAT' Assembler performs the translation with only one pass over the source deck, assembling the absolute instructions directly into core memory without the use of an intermediate "scratch tape". Instructions from the source program are (normally) assembled into the core locations from which they will subsequently be executed; at present there is no provision for automatic relocation. As each card image of the source program is processed, its image is listed on the printer along with the core location into which the corresponding binary instruction is assembled.

The 'THAT' language is "symbolic", meaning that symbols may be used for machine addresses and mnemonic names may be used for operation codes. Since it operates in a single pass, the 'THAT' Assembler will encounter address fields which contain symbols which have not yet been defined. The Assembler keeps lists of all such occurrences of undefined address symbols, and when the symbol is subsequently defined all references to it are properly "fixed up" in the assembled instructions in core memory. There are some important

restrictions on the use of such undefined symbols, however, in particular, a symbol which has not yet been defined cannot be used in a general address expression including any arithmetic operations at assembly time (for example: $X1 + 2$, or $2 * L3 + K0$).

The index field of a 'THAT' statement is further restricted: all symbols, whether used alone or in assembly - time expressions, must be defined before the index field is encountered. There is no provision for "fixing up" undefined symbols used in the index field.

In general, each line of 'THAT' code includes an operation code, either in absolute octal form or (more frequently) as a three-letter mnemonic. These mnemonics must be one of the following:

- (1) A standard G-20 machine language opcode mnemonic, as listed in the G-20 Reference Manual and in Appendix of this manual; or
- (2) A "sudo" (pseudo-instruction) mnemonic. A sudo does not stand for an actual machine command but is rather an instruction to the 'THAT' assembler, to be executed when the sudo is encountered during the assembly process. All 'THAT' sudos are listed in alphabetical order and explained in Chapter 3 of this manual.

1.1 SYMBOLS

The purpose of symbols is three-fold: (1) The programmer can refer symbolically to addresses which will not be known until the entire program has been written and assembled. (2) The programmer can parametrize his program and assign values to the parameters at assembly time, so that sizes of buffers, data storage blocks, program segments, etc., can subsequently be changed by simple reassembly runs. (3) The symbols can give some mnemonic value to the program, aiding the programmer in the task of writing, debugging, and changing the program.

Each 'THAT' symbol has the form of a class name followed by an integer; the integer is referred to as the "subscript" part of the symbol. Class names are one character, and can be any of the 26 letters or one of the four special characters: \neg , \leftarrow , \rightarrow , or $|$. These rules are summarized by the following Backus Normal Form:

```

<class name> ::= <letter> |  $\neg$  |  $\leftarrow$  |  $\rightarrow$  | <the mark '| '>
<subscript>  ::= <integer> | <empty>
<symbol>    ::= <class name> <subscript>

```

Notice that the subscript can be omitted; this has the same meaning as a zero subscript.

Examples:

```

L4
 $\neg$ 27
|3
T    (same as: T0)

```

The possible symbols are divided into 30 classes by the class names. All symbols of a particular class will be either:

- (1) Label symbols, whose values can be defined independently and in any order; or
- (2) Regional symbols, all referring to the same region and all bearing a fixed relationship to each other.

These two kinds are discussed in sections 1.1.1 and 1.1.2, below. The one class name 'A' has special significance, and is discussed in section 1.1.3.

Symbols are most frequently used to represent addresses with values between 0 and $2^{16} - 1$. However, a symbol may be defined (by a 'DEF' sudo) to have any value between 0 and $2^{30} - 1$.

1.1.1 LABEL SYMBOLS

All symbols with a particular class name can be declared to be label symbols with a 'LBL' ("LaBeL") sudo instruction. The 'LBL' declaration contains the class name character followed by the maximum subscript integer which labels of the class will be allowed. For example:

LBL K20

declares a set of 21 label symbols: K0, K1, K2, ..., K20. These symbols are free and arbitrary and can be defined in any order with any set of values. The symbols of the class are related only in that they occupy adjacent positions in the symbol table created by the Assembler. This fact may be of importance to the programmer who needs to reuse symbols or reclaim symbol table space during assembly of very large programs; see the sudo instructions CHK, LBL, and REL in Section 4 for more information. The maximum subscript given in the 'LBL' declaration is used by the Assembler to allocate symbol table space.

1.1.2 REGION SYMBOLS

A class name will denote a region if:

- (1) any symbol in that class is given a value (by a 'DEF' sudo instruction), and if
- (2) that class has not previously been declared to be label symbols (by a 'LBL' sudo instruction).

All symbols with a regional class name refer to the same region, and their values are related in a fixed way: the symbol whose subscript part is the integer n will stand for the n th memory address of the region. Thus, defining any one symbol of the class defines them all.

Example: assume that R has not appeared in a 'LBL' declaration; then:

DEF $R0 = 40$

will make R a region whose first cell is address 40 (an index register). Then all R symbols will be defined; e.g. $R9 = 49$, and in general $Rn = 40 + n$ is any integer constant. The following 'DEF' operation would have the same effect:

DEF $R9 = 49$

The expression: $R0 + 23$ is equivalent to the symbol: $R23$, if R is a region.

A class of symbols which has been used as a region can later be declared in a 'LBL' sudo instruction and thereafter be used as independent label symbols. Conversely, a class name which has been used for labels can be changed into a region if all labels of that class are undefined with a sudo instruction of the form:

REL <class name>

and if any one of the members of the class subsequently is defined in a DEF sudo.

1.1.3 THE 'A' SYMBOLS

The symbols in class 'A' have special significance in the 'THAT' language and cannot be used as label symbols. The symbol "A" or "A0" always has as value the current value of the Assembler's location counter, i.e. the memory location into which the current instruction is to be assembled. After processing each line of the source program, the Assembler increments the value of 'A' by the number of binary words it has loaded into memory. The 'A' value is printed on each line of the Assembly listing.

The 'A' symbols other than "A0" behave as if A were a region name; that is, A_n has the value: $A + n$.

Example:

TRA A + 3 (or: TRA A3)

has the same effect as:

	LBL	L2
	:	
L1	TRA	L1 + 3

1.2 EXPRESSIONS

Symbols may be used to build expressions, whose syntax can be defined in Backus Normal Form as follows:

```

<TERM> ::= <DEFINED SYMBOL> | <INTEGER> | <OCTAL INTEGER> | <POWER OF TWO>
<OCTAL DIGIT> ::= 0|1|2|3|4|5|6|7
<DIGIT> ::= <OCTAL DIGIT> | 8|9
<INTEGER> ::= <DIGIT> | <INTEGER> <DIGIT>
<OCTAL INTEGER> ::= /<OCTAL DIGIT> | <OCTAL INTEGER> <OCTAL DIGIT>
<POWER OF TWO> ::= $<INTEGER>
<OPERATOR> ::= + | - | *
<EXPRESSION> ::= <TERM> | <EXPRESSION> <OPERATOR> <TERM> | <EMPTY>

```

EXAMPLES:

```

418
/77 * $12
L1 -6-L0*/3

```

Here <DEFINED SYMBOL> means a symbol whose value has been defined previously in the assembly. The symbol may have been defined in any of the following ways:

- (1) It may be a regional symbol and therefore have received a value from the 'DEF' sudo which defined the region.
- (2) It may be a label symbol which has been explicitly defined by a 'DEF' sudo.
- (3) It may be a label symbol which has appeared in the location field of a previous instruction and therefore been defined with the value of 'A' for that instruction.

An expression defined by these rules can be used in the address and index fields of a line of 'THAT' code. The meaning of an expression is obtained by performing the indicated operations from left to right with no hierarchy and truncating to 32 bits after each operation. Thus, $2 + 3 * 4 = 20$.

The <TERM> "\$n", where n is an <INTEGER>, has the value $2 \uparrow n$; i.e., "\$n" stands for a 1-bit in bit position n of a logic word. An empty expression or term will have the value zero.

Expressions are generally used to represent G-20 (or G-21) addresses, and their values will therefore be positive integers less than 2^{16} . The rules for expressions outside this range are more complex, but are contained in the following paragraphs.

The value of an expression is generally computed in logic format, and will therefore be a positive integer between 0 and $2^{32} - 1$. The result of each arithmetic operation is shifted to zero exponent, truncated on both ends to 32 bits, and made positive (by an 'STL' command in the Assembler). There is an exception to the logic format, however; the right hand operand of each multiplication ('*') operation will be accessed numerically (by an 'MPY' command in the Assembler). Thus, the expression:

$\$24 * /377$

will be computed by 'THAT' to be $/377000000$; however, the expression with the operands are reversed:

$/377 * \$24$

will be computed to be 0 since the value $\$24$ will be accessed numerically. In the expression

$/3 * \$4 + 13 * \24

both $\$4$ and $\$24$ will be accessed numerically.

Although an expression is generally computed as a 32 bit logic word, the final result may be truncated to a smaller field, determined by the way that the expression is used. If an expression is used:

- (a) as the address of a G-20 command, it will be truncated to 16 bits, with the high-order bit stored as bit 30 of the command. See Section 2.6.
- (b) in the index field of a G-20 command, its value must be between 0 and 63 or an error message will be printed. See Section 2.7.
- (c) in a 'DEF' sudo, the expression will be truncated modulo 2^{30} (i.e., both flag bits will be set to 0). See Chapter 3.
- (d) in a 'LWD' or 'WRD' sudo, all 32 bits will be stored (EXCEPTION: 'WRD' sudo, if the expression is negative).

CHAPTER 2 - SOURCE PROGRAM FORMAT

2.0 SUMMARY OF FORMAT

A line of 'THAT' language source code contains information in some or all of the following fixed fields:

- | | |
|--------------|---------------------|
| 1. Language | - Columns 1 - 2 |
| 2. Location | - Columns 4 - 8 |
| 3. Flag | - Column 13 |
| 4. Operation | - Columns 15 - 17 |
| 5. Mode | - Column 20 |
| 6. Address | - Columns 24 - 67 |
| 7. Index | - Columns 24 - 67 |
| 8. Comments | - Columns |
| Immaterial | - All other Columns |

Example:

(Columns)

1	4	8	13	15	17	20	24
---	---	---	----	----	----	----	----

TH	E4		2	CLA	0	/77, R2; THIS IS A COMMENT		
↑	↑		↑	↑	↑	↑	↑	↑
1.	2.		3.	4.	5.	6.	7.	8.

2.1 LANGUAGE FIELD (Columns 1-2)

When card images are typed-in from a remote teletype, the language field is used to set the meaning of the TAB key for the language. The mnemonic 'TH' will set the TAB columns for 'THAT' card images as follows:

<u>Tab</u>	<u>Column</u>	<u>Field</u>
1	4	Label
2	15	Opcode
3	20	Mode
4	24	Address, Index Register
5	40	Comment

For more details, see SECTION 2 of the User's Manual.

2.2 LOCATION FIELD (Columns 4-8)

In general the location field will be blank unless a reference is made to that line of code. The location field may contain any of the following:

1. A label which is currently undefined. The effect is to define that label by giving it the current value of the location counter ('A').
2. An expression which equals the current value of the location counter. This can be used for explanatory or documentary purposes.
3. Any string of characters starting with the letter 'A'. The contents of the rest of the field will be ignored and can be used for a comment.

Examples:

	MPY		M5;	SHIFT RIGHT 5 OCTALS
	.			
	.			
	.			
M5	105		1;	SHIFT CONSTANT
	LXP	0	20, R2;	
E2	STZ		P0, R2;	ZERO A LOCATION IN MEMORY
	SXT	0	1, R2;	DECREMENT AND TEST
	TRA		E2;	LOOP
	.			
	.			
	.			

2.3 FLAG FIELD (Column 13)

The flag field is used to specify the flag bits of the word generated.

FLAG COLUMN	FLAG BIT(S) LOADED
0 OR BLANK	NONE
1	BIT 30
2	BIT 31
3	BITS 31 AND 30

Note that in the G-21 Central Processor, Flag Bit 30 has the special significance of the highest-order bit of the address. See Note 1, SECTION 4.1.3. The flag field is ignored on all sudo instruction cards, unless the sudo is 'ADC', 'LWD', or 'NAM'.

2.4 OPERATION FIELD (Columns 15-17)

The operation field may contain one of the following:

1. Blanks. The line will be processed as a 'COM' sudo, i.e., a comment card.
2. An octal integer (without the preceding slash). In this case, it will be interpreted as the operation part of a G-20 instruction and the octal integer will be right justified in bits 29 to 21 of the assembled instruction.
3. The three-letter mnemonic for G-20 operation. The corresponding octal code will be loaded as part of the assembled instruction. G-20 mnemonics are listed in the appendix.
4. The mnemonic for a 'THAT' sudo. The action taken for the possible sudos is described in Part 4.

2.5 MODE FIELD (Column 20)

Each G-20 mnemonic has associated with it a "normal" mode for that operation as described below. If the normal mode is desired, the mode field may be left empty; otherwise, 0, 1, 2 or 3 must be punched. A mode punch always supercedes the normal mode. The mode field of a sudo is ignored. (EXCEPTION: See 'LWD' sudo, 'ADC' sudo and 'NAM' sudo.) Section 3 contains a summary of the addressing modes.

All G-20 mnemonics are mode 2 except the following which are mode 0.

STI	STL	TRA	REP
STS	STZ	TRM	
STD			

2.6 ADDRESS FIELD (Columns 24-67)

The address field normally contains the operand or the address of the operand. Blanks in the address field are ignored (except in 'ALF' sudo and 'NAM' sudo).

The address is terminated by a comma, a semi-colon, or Column 68 (which is not scanned), whichever occurs first. If it is terminated by a comma, an index is then expected.

The following applies to the address field only if the operation field contains a G-20 mnemonic or an octal integer.

1. If it is blank, address (bits 14-0 and bit 30) of the assembled instruction will be zero.
2. If it is a single symbol which is already defined, the value of the symbol will be placed in the address (bits 14-0 and bit 30) of the assembled instruction. If the symbol is a label which is not yet defined, its value will be placed in the address when it is defined.
3. If it is an expression, the value of the expression will be entered as the address in the assembled instruction. All symbols in the expression must have been defined previously or an error message will be printed. See 5.1.1.

If the operation field contains a G-20 mnemonic or the 'ADC' sudo, the value of the corresponding expression must be less than 2^{16} and is converted to the 15 and 1 bit format by G-21 commands; i.e., if bit 15 is non-zero, bit 30 is set to one and bit 15 is set to zero.

EXAMPLES:

```

DEF      A = /120000, P0 = /124000, R0 = /40;
CLA  3  P0, R0;
      .
      .
      .
DEF      A = P0;
LWD      P64;
```

After these cards are assembled, location /120000 contains /1 605 40 24000 (Note that bit 15 is shifted to bit 30). Location /124000 contains /0 000 01 24100.

2.7 INDEX FIELD (Columns 24-67)

If any index register is used, the address field must be terminated by a comma, followed by a symbol or an expression whose value is the number of an index register. Blanks in the index field are ignored, and the field is terminated by a semi-colon or Column 68 (which is not scanned), whichever occurs first.

The value of the expression in the index field is loaded right-justified into bits 20-15 of the assembled instruction; if the value is not defined, an error message will be printed. If the operation field contains a G-20 mnemonic, an error message will be printed in the value of the index field is greater than 63.

2.8 COMMENT FIELD (Columns 24-80)

All columns to the right of the first semi-colon in the address-index field are ignored by the Assembler, and may therefore be used for comments. Comments may extend to Column 80. All columns of the input line including the 'AND' sequence number are printed (unless assembly printing has been turned off).

CHAPTER 3 - SUDO INSTRUCTIONS IN 'THAT'

3.0 INTRODUCTION

A sudo (pseudo-instruction) is an instruction to 'THAT' rather than a G-20 command to be assembled for later execution. The mnemonic name of the sudo is punched in the operation field of the source program card.

For all sudos the following holds:

1. The Location field is first treated as described in Section 2.2 for machine commands.
2. The Flag and Mode fields are ignored (EXCEPTIONS: 'LWD' sudo, 'NAM' sudo and 'ADC' sudo.)
3. Thereafter, the specific action for the particular sudo takes place.
4. A sudo may be listable or non-listable: the parameter set given by the address field of a listable sudo may be repeated, separated by commas, as many times as desired in the space provided on the card up to Column 67, while only one parameter set is allowed in the address field of a non-listable sudo. The effect of a listable sudo is the same as if the sudo was repeated on successive lines with one parameter set per line; the parameter sets are processed in the left-to-right order.

Section 3.1 contains a reference list of all sudos in 'THAT'. The remainder of Chapter 3 consists of an alphabetical listing of the sudos, with an explanation and examples of the use of each one.

The format used in explaining the sudos is as follows:

```

XXX      EXPRESSION
                                LISTABLE
      'EXECUTE EXTRA EXEC'
```

The first line gives the three letter sudo name and the type and format of the parameter set (s). The second line states whether the sudo is listable or non-listable for sudos for which the concept is meaningful. The third line contains a word or more describing the action of the sudo. (NOTE: the above sudo is only a hypothetical example.)

ADC <EXPRESSION>|<EXPRESSION>,<INDEX>
 NON-LISTABLE
 'ADDRESS CONSTANT'

THE FUNCTION OF 'ADC' IS THE SAME AS THE G-20 MNEMONIC 'OCA', EXCEPT THE NORMAL MODE IS ZERO RATHER THAN TWO. 'ADC', USED FOR ADDRESS CONSTANTS, MAY HAVE AN ADDRESS WHICH MUST BE LESS THAN 2^{16} AND MAY ALSO HAVE AN INDEX. 'ADC' IS THE SUDO WHICH WOULD NORMALLY BE USED WHEN COMMANDS ARE TO BE ASSEMBLED AT EXECUTION TIME.

EXAMPLES:

```

ADC      /177777;
1 OCA 0  /77777;
```

THESE TWO INSTRUCTIONS ARE EQUIVALENT.

ALF <BLANK> <STRING>|<DIGIT> <STRING>
 NON-LISTABLE
 'ALPHANUMERIC'

THE EFFECT IS TO LOAD THE G-20 INTERNAL REPRESENTATION OF THE STRING OF CHARACTERS INTO SUCCESSIVE MACHINE LOCATIONS, 4 CHARACTERS PER WORD. THE DIGIT GIVES THE NUMBER OF WORDS TO BE LOADED, WITH A BLANK BEING TREATED AS 1, AND 0 BEING TREATED AS 10. THE BLANK OR DIGIT MUST APPEAR IN THE FIRST POSITION OF THE ADDRESS FIELD, COLUMN 24. THE STRING TO BE LOADED EXTENDS FROM COLUMN 25 TO COLUMN $(24+4K)$, WHERE K IS THE NUMBER OF WORDS SPECIFIED.

EXAMPLES:

```

W1  ALF      4ERROR NUMBER ONE
```

THIS LINE WILL CAUSE THE LOADING OF

ERRO	INTO	W1
R NU	INTO	W1+1
MBER	INTO	W1+2
ONE	INTO	W1+3

THIS IS EQUIVALENT TO

```

W1  ALF      1ERRO
    ALF      R NU
    ALF      MBER
    ALF      1 ONE
```

CHK <SYMBOL>

LISTABLE

CHECK

THE FUNCTION IS TO CHECK WHETHER OR NOT LABELS WHICH HAVE BEEN USED ARE DEFINED. THE SYMBOL MUST BE A LABEL. IF ITS SUBSCRIPT IS ZERO OR BLANK, THEN THE SUBSCRIPT IS CONSIDERED TO BE THE MAXIMUM ALLOWED SUBSCRIPT. THE LABELS FROM <IDENTIFIER>0 TO <IDENTIFIER>SUBSCRIPT ARE THEN CHECKED TO SEE IF ALL THOSE WHICH HAVE BEEN USED ARE DEFINED. IN CASE AN UNDEFINED LABEL IS ENCOUNTERED, AN ERROR PRINT OUT TAKES PLACE WITH THE FOLLOWING FORM:

UND T5 26347 54362

THIS MEANS THAT THE LABEL T5 IS UNDEFINED, AND THAT IT HAS LAST BEEN USED IN LOCATION /26347 AS A 16-BIT CONSTANT AND IN LOCATION /54362 AS AN ADDRESS TO AN INSTRUCTION.

THE CHECKING WILL CONTINUE UNTIL ALL THE SUDO-PARAMETERS HAVE BEEN EXHAUSTED.

EXAMPLES:

```

LBL      D5;
LBL      W10;
LBL      R90;
( PROGRAM )
CHK      D,W5,R;

```

ALL OF THE D'S AND R'S AND W0 TO W5 ARE CHECKED.

COM <IMMATERIAL>

COMMENT

THE REST OF THE LINE IS IGNORED.

EXAMPLES:

```

      LBL      L1;
      COM      THIS IS A COMMENT
      DEF      A=/30000;
L1     COM      GEE... ANOTHER COMMENT

```

THESE LINES WILL BE PRINTED. TWO L'S WILL BE DECLARED AS LABELS AND L1 WILL BE GIVEN THE VALUE /30000. HOWEVER, NO CODE WILL BE COMPILED.

CPY <EXPRESSION>,<EXPRESSION>
 LISTABLE

COPY

LET THE VALUE OF THE FIRST AND THE SECOND EXPRESSIONS BE N1 AND N2, RESPECTIVELY.

THE NEXT N1 WORDS WILL BE FILLED BY COPYING FROM THE LAST N2 WORDS ASSEMBLED. THAT IS, THE WORDS IN A-N2, A-N2+1,.....A-1 WILL BE COPIED REPEATEDLY UNTIL N1 HAVE BEEN COPIED. N1 NEED NOT BE A MULTIPLE OF N2; IF N1 = 0, NO WORDS WILL BE COPIED.

AFTER 'CPY' HAS BEEN EXECUTED, THE LOCATION COUNTER 'A' HAS BEEN INCREASED BY N1.

WARNING: IF THE LAST N2 WORDS CONTAIN ANY UNDEFINED LABELS, THESE WILL NOT LATER BE DEFINED IN THE COPIES.

EXAMPLES:

```

W8     LWD      /737
      LWD      W53;
      CPY      500,2

```

(W8) AND (W8+1) WILL BE COPIED INTO THE NEXT 500 LOCATIONS.

```

E1     LWD      0;
      CPY      499,1;

```

THE EFFECT IS TO CLEAR 500 LOCATIONS STARTING AT E1.

DBG

'DEBUG'

THE FUNCTION IS TO TURN ON THE SELECTIVE TRACE SWITCH IN MONITOR. IN RUNNING THE PROGRAM, ALL COMMANDS WITH A 2 FLAG (A 1 IN BIT 31) WILL BE LISTED ON THE PRINTER IN THE FORMAT FOR MONITOR TRACE DESCRIBED IN THE APPENDIX.

A 'DBG' SUDO CARD MAY BE PLACED ANYWHERE IN THE 'THAT' DECK. COMMANDS MAY BE MARKED FOR TRACING EITHER BY INSERTING A 'FLG' SUDO BEFORE, OR PUNCHING A 2 IN THE FLAG FIELD (COLUMN 13) OF THE CARD WHOSE INSTRUCTION IS TO BE TRACED.

DEC

<IMMATERIAL>

'DECIMAL LISTING'

THE FUNCTION IS TO CAUSE SUBSEQUENT CONVERSION FOR PRINTING OF THE CURRENT INSTRUCTION COUNTER AND REGION AND LABEL ADDRESSES TO BE DONE IN DECIMAL.

EXAMPLES:

```

DEF      A=/20000;
DEC      PRINT IN DECIMAL
RGN      A;
A0      8192

```

NOTICE THAT THE REGIONAL SYMBOL IS CONVERTED IN DECIMAL.


```
DEF      <SYMBOL>=<EXPRESSION>
                                LISTABLE
'DEFINE'
```

THE VALUE OF THE EXPRESSION WILL BE CALCULATED AND TAKEN MODULO 2+30, AND THE SYMBOL WILL BE GIVEN THIS VALUE.

IF THE LETTER OF THE SYMBOL HAS BEEN DECLARED AS A LABEL, THE PARTICULAR LABEL GIVEN IS THEREBY DEFINED. IF THE LETTER IS NOT A LABEL, THE CORRESPONDING REGIONAL BASE IS DEFINED AS

◀ EXPRESSION ▶ - ◀ SUBSCRIPT ▶

WHERE THE SUBSCRIPT NORMALLY EQUALS ZERO.

EXAMPLES:

DEF A=13000

THE MEMORY LOCATION FOR THE NEXT INSTRUCTION IS /13000.

LBL B30
DEF B0=/22750

THIRTY ONE B'S ARE DESIGNATED AS LABELS, AND B0 IS GIVEN THE VALUE /22750. B1, B2, . . . , B30 ARE UNDEFINED.

```
DEF      C10=17000;
```

C0 IS GIVEN THE VALUE /6766, AND ALL C'S ARE DEFINED.

[illegible]

THE EFFECT IS TO GIVE AN OCTAL DUMP ON THE PRINTER OF THE LOCATIONS FROM THE VALUE OF THE FIRST EXPRESSION UP TO AND INCLUDING THE VALUE OF THE SECOND EXPRESSION.

WARNING: THERE IS NO CHECK THAT THE VALUES ARE PROPER
MACHINE LOCATIONS.

EXAMPLES:

DMP /21000,/22000

AN OCTAL DUMP WILL BE GIVEN FROM LOCATION /21000 UP TO AND INCLUDING THE LOCATION /22000.

DMP A-100,A-1;

AN OCTAL DUMP OF THE LAST 100 LOCATIONS WILL BE GIVEN.

ENT <IMMATERIAL>

'ENTRY'

THE EFFECT IS TO UPSPACE THE PRINTER TWICE (IF THE PRINTING IS ON), AND ASSEMBLE AN ALL ZERO WORD. THIS SUDDO CAN BE USED FOR ENTRY INTO A SUBROUTINE. A LABEL APPEARING IN THE LOCATION FIELD WILL BE DEFINED AS USUAL.

EXAMPLES:

P1 ENT SUBROUTINE

THIS DESIGNATES THE ENTRY INTO A SUBROUTINE THAT IS REFERRED TO BY THE LABEL P1. ZERO IS LOADED INTO THE LOCATION P1.

FLG <BLANK>

'FLAG'

THE FUNCTION IS TO INSERT A 2 FLAG (BIT 31) IN THE NEXT G-20 INSTRUCTION STORED. BECAUSE OPERAND ASSEMBLY (OA) COMMANDS ARE NOT TRACED, PLACING A 'FLG' SUDDO BEFORE AN 'OA' COMMAND CAUSES THE NEXT NON-'OA' COMMAND TO BE TRACED.

EXAMPLES:

```

DBG
FLG
P1  CAL      D,I;
    FLG
    OCA      I1;
P2  STL      C,I2;

```

THE COMMANDS LABELED P1 AND P2 WILL BE TRACED.

[illegible]

THE FUNCTION IS TO LOAD THE OCTAL REPRESENTATION OF THE DECIMAL NUMBER INTO THE NEXT TWO LOCATIONS.

WARNING: THE ABSOLUTE VALUE OF THE NUMBER MUST BE LESS THAN 3.450873173389₁₀⁶⁹ AND THE EXPONENT LESS THAN 70, OR AN EXPONENT OVERFLOW WILL OCCUR AT ASSEMBLY TIME.

EXAMPLES:

W10	FPC	$10,4.000159_{10}16$
W11	FPC	$-2_{10}5,3.44463_{10}-5$

W10 AND W10+1 WILL BE LOADED WITH 10, W10+2 AND W10+3 WILL BE LOADED WITH $4.000159 \times 10^{+16}$, W11 AND W11+1 WILL BE LOADED WITH $-2 \times 10^{+5}$, AND W11+2 AND W11+3 WILL BE LOADED WITH $3.44463 \times 10^{+5}$, ALL IN STANDARD G-20 FULL PRECISION FORM. W10 AND W11 MUST BE LABELS, SINCE THEY ARE NOT IN ADJACENT LOCATIONS.

[illegible]

THE FUNCTION IS TO LOAD THE OCTAL REPRESENTATION OF THE DECIMAL NUMBER INTO THE NEXT LOCATION. THE MANTISSA OF THE NUMBER IS ROUNDED TO SEVEN (OCTAL) DIGITS BEFORE STORING.

EXAMPLES:

W12	HPC	0, 1, 2, 3;
	HPC	-4.15 ₁₀ -6;

0, 1, 2, 3, AND -4.15×10^{-6} WILL BE LOADED INTO FOUR
CONSECUTIVE LOCATIONS STARTING AT W12.

WARNING: IT IS A DETECTABLE ERROR IF A NUMBER WHOSE ABSOLUTE VALUE IS GREATER THAN $(8+7-1)*8+63$ IS PUNCHED.

LBL

<SYMBOL>

LISTABLE

'LABEL'

THE LETTER IS DECLARED TO BE A LABEL. IF THE LETTER HAS NOT PREVIOUSLY APPEARED IN A 'LBL' SUDO, THEN THE SUBSCRIPT IS THE MAXIMUM SUBSCRIPT WHICH MAY BE USED FOR THAT LABEL.

IF THE LETTER HAS PREVIOUSLY APPEARED IN A 'LBL' SUDO, ITS NEW SUBSCRIPT MAY NOT BE GREATER THAN THE SUBSCRIPT FIRST DESIGNATED UNLESS THE LABEL HAS FIRST BEEN RELEASED BY THE 'REL' SUDO OR ELSE IT IS AN ERROR. THE FOLLOWING ACTIONS TAKE PLACE:

FIRST, THE OPERATION OF A 'CHK' SUDO IS DONE ON THE SYMBOL. THEN THE LABELS FROM <LETTER>0 TO <LETTER><SUBSCRIPT> ARE CLEARED TO USE AGAIN, WHILE ANY LABELS GREATER THAN THE SUBSCRIPT APPEARING IN <LETTER><SUBSCRIPT> ARE LEFT UNTOUCHED.

IN CASE 'CHK' FINDS ONE OR MORE UNDEFINED LABELS AN ERROR MESSAGE WILL BE PRINTED (SEE 'CHK') AND THE VALUE OF THE LABEL WILL BE CLEARED FOR REDEFINITION.

EXAMPLES:

```
LBL      D10
```

DO THROUGH D10 WILL BE PERMITTED FOR USE AS LABELS.

```
( PROGRAM )
LBL      D7
( PROGRAM )
```

THE LABELS D0 THROUGH D7 WILL BE CLEARED FOR REDEFINITION AS NEW LABELS, WITH AN ALARM MESSAGE PRINTED IF ANY ARE UNDEFINED.

LIN <EXPRESSION>

NON-LISTABLE
CARD IMAGE NOT PRINTED

'LINE'

THE FUNCTION IS TO UPSPACE THE PRINTER
N=<EXPRESSION> LINES, IF PRINTING IS ON.
IF N = 0 OR THE ADDRESS FIELD IS BLANK, 1 LINE UPSPACE
WILL OCCUR.

EXAMPLES:

```
CLA      P9;
LIN      2;
EXL      K21;
```

ABOVE ARE THE CARDS AS THEY WERE PUNCHED. BELOW IS
THE COMPILATION OF THE CARDS.

```
CLA      P9;

EXL      K21;
```

NOTICE THAT 2 LINES WERE SKIPPED AND THE 'LIN'
SUDDO WAS NOT PRINTED.

LWD <EXPRESSION>|<EXPRESSION>,<EXPRESSION>
LISTABLE

'LOGIC WORD'

THE EFFECT IS TO LOAD THE VALUE OF THE EXPRESSION
INTO THE NEXT MACHINE LOCATION AS A LOGIC WORD
(I.E. WITH AN 'STL' COMMAND). ANY PUNCHING IN THE
FLAG OR MODE FIELD WILL TAKE PRECEDENCE OVER THE INFORMATION
THAT WOULD OTHERWISE BE LOADED INTO BITS 28 TO 31.

IF THE ADDRESS EXPRESSION IS TERMINATED BY A
COMMA AN INDEX REGISTER IS EXPECTED AND WILL TAKE
PRECEDENCE OVER THE INFORMATION IN BITS 15 TO 20.

NO CHECKS ARE MADE TO SEE IF THE VALUES OF THE
EXPRESSIONS ARE WITHIN THE LIMIT OF THE FIELDS.

EXAMPLES:

```
DEF      A=/20000;
LBL      E2;
20000 E0  LWD      /350 + $4;
20001 E1  LWD      1  /7777+$1;
20002 E2  LWD      /777777777777;
```

THE VALUE OF /350+\$4 = /370 WILL BE LOADED INTO
LOCATION 20000 (E0). /7777+\$1+ BIT 28 (MODE 1 PUNCH) =
/2000010001 WILL BE LOADED INTO LOCATION 20001 (E1), AND
/377777777777 WILL BE LOADED INTO LOCATION 20002 (E2).

MTT <EXPRESSION>

NON-LISTABLE
PRINTING BEFORE EXECUTION

'MARK TRANSFER TO'

THE FUNCTION IS TO CHECK THAT NO ASSEMBLY ERRORS HAVE OCCURRED AND THAT ALL USED LABELS ARE CURRENTLY DEFINED. IF NO ERRORS ARE DETECTED, 'THAT' EXECUTES A 'TRM' TO THE LOCATION DEFINED BY <EXPRESSION>. THE ROUTINE MAY RETURN THROUGH ITS MARK IF ASSEMBLY IS TO CONTINUE OR TRANSFER CONTROL TO THE MONITOR BY EXECUTING THE MONITOR HALT ROUTINE.

EXAMPLE:

```

E1  ENT      ZERO DATA REGION
     LXP      0   100,R1;
E2  STZ      D,R1;
     SXT      0   1,R1;
     TRA      E2;
     TRA      1   E1;
     CHK      E;
     MTT      E1;

```

THE SIX INSTRUCTIONS STARTING AT E1 WILL BE ASSEMBLED AND EXECUTED. ASSEMBLY WILL THEN CONTINUE.

NAM <STRING>

NON-LISTABLE

'NAME'

THE EFFECT IS TO PACK THE SIX BIT REPRESENTATION OF THE 5 CHARACTERS IN COLUMNS 24 TO 28 INTO THE RIGHTMOST 30 BITS OF THE NEXT MACHINE LOCATION. ANY PUNCHING IN THE FLAG OR MODE FIELD WILL TAKE PRECEDENCE OVER THE INFORMATION THAT OTHERWISE WOULD HAVE BEEN LOADED INTO BITS 28 TO 31.

EXAMPLES:

NAM PN3.\$

THE 6 -BIT REPRESENTATIONS OF THE CHARACTERS P, N, 3, . AND \$ WILL BE LOADED INTO THE NEXT MACHINE LOCATION. THIS IS THE SAME AS

LWD /20 16 43 53 65;

OCT <IMMATERIAL>

OCTAL LISTING

THE FUNCTION IS TO CAUSE SUBSEQUENT CONVERSION FOR PRINTING OF THE CURRENT INSTRUCTION COUNTER AND REGION AND LABEL ADDRESSES TO BE DONE IN OCTAL. (SEE *DEC*).

OCT IS ASSUMED WHEN ASSEMBLY BEGINS.

EXAMPLES:

```

DEF      A=8192;
OCT      PRINT IN OCTAL;
RGN      A;
A0      20000

```

NOTICE THAT THE REGIONAL SYMBOL IS CONVERTED TO OCTAL.

OPM <IMMATERIAL>

OPERATOR MESSAGE

THE FUNCTION IS TO PRINT THE CURRENT TIME, DATE AND OPERATOR INFORMATION. COLUMNS 24 TO 80 OF THE INSTRUCTION CARD ARE REPLACED BY THE ABOVE INFORMATION.

EXAMPLES:

```

OPM      OPERATORDI01 01 JUL 64      23S S
           ↑      ↑
           OPERATOR  DATE            TIME

```

THIS IS THE LISTING WITH THE OPERATOR MESSAGE.

OUT <EXPRESSION>

NON-LISTABLE
PRINTING BEFORE EXECUTION

OUT

THE EFFECT OF THE *OUT* SUDO IS TO CHECK THAT NO ASSEMBLY ERRORS HAVE OCCURED AND THAT ALL USED LABELS ARE DEFINED. IF NO ERRORS ARE DETECTED, *THAT* EXECUTES A *TRA* TO THE LOCATION DEFINED BY <EXPRESSION>.

IT IS A DETECTABLE ERROR IF THIS LOCATION IS NOT A VALID MACHINE ADDRESS.

EXAMPLES:

```
( PROGRAM )
OUT      E1
```

CONTROL WILL BE TRANSFERRED TO LOCATION E1. EITHER THIS SUDO OR *MTT* IS NORMALLY USED TO START EXECUTION OF A PROGRAM.

PAG <IMMATERIAL>

PRINTING AFTER EXECUTION

PAGE

IF PRINTING IS TURNED ON, THE PAPER IN THE PRINTER WILL BE MOVED TO THE NEXT PAGE.

PBC <EXPRESSION>,<EXPRESSION>|
 <EXPRESSION>,<EXPRESSION>,1NON-LISTABLE
PRINTING BEFORE EXECUTION

PUNCH BINARY CARDS

THE FUNCTION IS TO PUNCH A ROW-BINARY DECK OF THE MEMORY LOCATIONS FROM THE ADDRESS GIVEN BY THE FIRST EXPRESSION UP TO AND INCLUDING THE ADDRESS GIVEN BY THE SECOND EXPRESSION. IF A THIRD PARAMETER '1' APPEARS, THE SYMBOL TABLE WILL BE PUNCHED. THUS IT IS POSSIBLE LATER TO ADD TO OR TO CORRECT A PROGRAM WITH THE USE OF SYMBOLS AFTER LOADING THE ROW-BINARY DECK (BY MEANS OF THE *RBC* SUDO OR THE MONITOR BAR ROUTINE).

WARNING: THERE IS NO CHECK ON THE VALUES BEING PROPER MACHINE LOCATIONS.

EXAMPLES:

PBC /20000.W53.1

A ROW-BINARY DECK WILL BE PUNCHED FROM LOCATION /20000 TO THE LOCATION OF W53. THE SYMBOL TABLE WILL ALSO BE PUNCHED.

PRT <SYMBOL>

LISTABLE
PRINTING BEFORE EXECUTION

'PRINT'

THE FUNCTION IS SIMILAR TO 'CHK', BUT IN ADDITION, IF THE PRINTING IS ON, THE VALUES OF ALL USED LABELS WILL BE LISTED ON THE PRINTER.

EXAMPLES:

PRT W, P, D, Q10;

ALL THE USED LABELS OF THE SYMBOLS W, P, D AND Q0 TO Q10 AND THE LOCATIONS TO WHICH THEY HAVE BEEN ASSIGNED ARE LISTED ON THE PRINTER.

RBC <EMPTY>

NON-LISTABLE
PRINTING BEFORE EXECUTION

'READ BINARY CARDS'

THE FUNCTION IS TO LOAD A ROW-BINARY DECK AS PREPARED BY THE 'PBC'-SUDO (EITHER WITH OR WITHOUT THE SYMBOL TABLE). THE ROW BINARY DECK SHOULD FOLLOW IMMEDIATELY WITH NO BLANK CARDS PRECEEDING IT. TWO BLANK CARDS SHOULD BE PLACED AT THE END OF THE BINARY DECK BEFORE THE REMAINING 'THAT' CARDS.

THE PROGRAM PORTION OF THE 'PBC'-DECK IS READ INTO THE SAME MACHINE LOCATIONS FROM WHICH IT WAS PUNCHED AND THE SYMBOL TABLE PORTION (IF PRESENT) IS READ INTO THE 'THAT' SYMBOL TABLE REPLACING THE SYMBOL TABLE BEING USED PRIOR TO THE 'RBC' SUDO.

NOTE: THIS CANNOT BE DONE FROM 'AND' FILES.

REL <SYMBOL>

LISTABLE

'RELEASE'

THE FUNCTION IS TO RELEASE LABELS; I.E., TO CLEAR THE DEFINITION OF A LETTER AS A LABEL SO THAT IT CAN BE USED THEREAFTER AS A REGION (OR A NEW LABEL).

FIRST 'CHK' IS PERFORMED. IF NO UNDEFINED LABEL IS ENCOUNTERED, THE LETTER IS THEN MARKED AS UNUSED. UNDER CERTAIN CIRCUMSTANCES THE SPACE USED FOR THE LABEL TABLE WILL ALSO BE RELEASED. THIS WILL OCCUR IF THE LETTER BEING RELEASED IS THE LAST LETTER DECLARED AS A LABEL, OR IF ALL LETTERS DECLARED SINCE HAVE BEEN RELEASED AND THEIR SPACE RECLAIMED.

IF AN UNDEFINED LABEL IS ENCOUNTERED BY 'CHK', AN ERROR MESSAGE WILL BE PRINTED (SEE 'CHK') AND THE ERROR IGNORED.

EXAMPLES:

```

LBL      R10
( PROGRAM )
REL      R
LBL      R11

```

THE SET OF LABELS R0 THROUGH R10 IS RELEASED AND THEN A NEW SET OF LABELS R0 THROUGH R11 IS DEFINED.

RET <IMMATERIAL>

PRINTING BEFORE EXECUTION

'RETURN'

THIS SUDO WILL EFFECT A RETURN TO THE LOCATION MARKED AS THE LAST CALL OF 'THAT'.

EXAMPLES:

```

RET      EXIT FROM THAT

```

CONTROL WILL RETURN TO THE PROGRAM WHICH CALLED 'THAT' AS A SUBROUTINE (USUALLY THIS WILL BE THE MONITOR).

RGN <SYMBOL> LISTABLE
 'PRINT REGIONAL SYMBOL'

THE FUNCTION OF THE 'RGN' SUDO IS TO CHECK THAT
 THE <SYMBOL> IS A DEFINED REGIONAL SYMBOL.
 IF THE PRINTING IS ON THE VALUE OF THE SYMBOL IS PRINTED
 AS IN 'PRT'.

```
DEF      P=/20000;
RGN      P201;
```

P201 20311

P201 IS A DEFINED REGIONAL SYMBOL-LOCATION 20311.

SXX <EXPRESSION> NON-LISTABLE
 'SET STORAGE EXTRACTOR'

THE VALUE OF THE EXPRESSION WILL BE STORED AS THE
 INTERNAL STORAGE EXTRACTOR IN 'THAT'. NORMALLY THE
 STORAGE EXTRACTOR IS 0.

WHENEVER A WORD IS STORED BY 'THAT' IT IS DONE AS
 FOLLOWS:

```
CAL   3   INSTRUCTION COUNTER (A)
EXL           STORAGE EXTRACTOR
ADL           WORD TO BE STORED
STL   1   INSTRUCTION COUNTER (A)
```

THUS 'SXX' CAN BE USED WHEN PARTS OF ALREADY LOADED
 WORDS HAVE TO BE CHANGED.

EXAMPLES:

```
DEF      A=/10000
E1  LWD   /17777;
    LWD   /17373
    LWD   /17313
    SXX   / 777
    DEF   A=E1;
    LWD   /24000
    LWD   /35000
    LWD   /71000
    SXX   0
```

FIRST THE INITIAL LOCATION IS DEFINED. THEN THE LOGIC WORDS /17777, /17373, AND /17313 ARE LOADED INTO THE FIRST THREE LOCATIONS, AND THE STORAGE EXTRACTOR IS SET TO / 777. THE LOCATION IS AGAIN GIVEN AS E1 FOR THE LOADING OF /24000, /35000, AND /71000. THUS THE VALUES /24777, /35373, AND /71313 WILL BE STORED IN LOCATIONS E1, E1+1 AND E1+2. THE LAST LINE WILL RESET THE STORAGE EXTRACTOR TO ZERO.

TOP <EXPRESSION>

PRINTING AFTER EXECUTION

'TYPE OR PRINT'

THE EXPRESSION, WHICH IS TAKEN MODULO 2, DETERMINES WHETHER OR NOT THE INPUT LINES WILL BE LISTED ON THE LINE PRINTER, AS FOLLOWS:

0: PRINTING OFF
1: PRINTING ON

WHEN PRINTING IS OFF ALL ACTION INVOLVING THE PRINTER WILL BE BYPASSED EXCEPT ERROR MESSAGE PRINTOUT.

WRD <SIGNED EXPRESSION>

LISTABLE

'WORD'

THE EFFECT IS TO STORE THE VALUE OF THE EXPRESSION INTO THE CORE LOCATION SPECIFIED BY THE LOCATION COUNTER 'A'. IF THE VALUE OF THE EXPRESSION IS NEGATIVE, 'WRD' WILL STORE IT INTO MEMORY AS AN INTEGER (I.E. WITH AN 'STI' COMMAND); IF POSITIVE, IT WILL BE STORED AS A LOGIC WORD (I.E. WITH AN 'STL' COMMAND).

EXAMPLES:

W8 WRD -/735+8

W8 WILL BE LOADED WITH THE NEGATIVE INTEGER /725

W10 WRD /7777777777

W10 WILL BE LOADED WITH THE LOGIC WORD /3777777777.

4.1 ERROR DETECTED DURING COMPILATION

ANY ERROR DETECTED BY 'THAT' DURING THE PROCESSING OF A LINE WILL CAUSE A PRINT OUT OF THE LINE OF CODE FOLLOWED BY AN ERROR MESSAGE, AS FOLLOWS.

4.1.1 ERRORS IN G-20 INSTRUCTIONS

AD U	UNDEFINED CONSTRUCTION IN ADDRESS FIELD OF G-20 INSTRUCTION
AD >	THE VALUE OF THE EXPRESSION IN THE ADDRESS OF A G-20 INSTRUCTION WITH MNEMONIC OPERATION IS NOT LESS THAN 2×16 .
FLAG	ERROR IN THE FLAG FIELD OF A G-20 INSTRUCTION
IR U	UNDEFINED CONSTRUCTION IN INDEX FIELD OF A G-20 INSTRUCTION
IR >	VALUE OF THE EXPRESSION IN INDEX OF A G-20 INSTRUCTION WITH MNEMONIC OPERATION IS NOT LESS THAN 64.
LABL	ERROR IN ENTRY FIELD
MODE	ERROR IN THE MODE FIELD OF A G-20 INSTRUCTION
OPER	ERROR IN OPERATION FIELD

4.1.2 ERRORS IN SUDO INSTRUCTIONS

AD U UNDEFINED CONSTRUCTION WHERE AN EXPRESSION IS
 NEEDED IN THE ADDRESS FIELD OF A SUDO.

A U 'A' IS NOT WITHIN BOUNDS OF MEMORY. (UPON STORING A WORD)

FLAG ERROR IN PARAMETER TO 'FLG' SUDO

LBL> A SUBSCRIPT ON A LABEL SYMBOL IS GREATER THAN ALLOWED

TERM UNDEFINED CONSTRUCTION WHERE A SYMBOL IS WANTED IN
 THE ADDRESS FIELD OF A SUDO.

WHAT A LETTER WHICH HAS NOT BEEN DECLARED AS A LABEL
 APPEARS IN A SYMBOL IN THE ADDRESS FIELD OF A SUDO
 WHERE A LABEL SYMBOL IS REQUIRED.

4.1.3 NOTES

NOTE 1 A ONE FLAG OR A THREE FLAG HAS BEEN PUNCHED IN
 FLAG COLUMN OF A G-20 MNEMONIC AND HAS ALTERED
 THE EFFECTIVE OPERAND ADDRESS OF THE INSTRUCTION.

NOTE 2 AN 'XEQ' OPCODE HAS BEEN PUNCHED WITH A MODE
 0 OR 1 AND WILL CAUSE AN OPCODE FAULT IF EXECUTED.

ADDRESS-OPCODE FAULT:

AN ADDRESS OPCODE FAULT IS DETECTED WHENEVER THE G-20 ATTEMPTS TO PROCESS A COMMAND INSTRUCTION IN WHICH ONE OR MORE OF THE FOLLOWING OCCURS:

1. THE BIT CONFIGURATION IN THE OPERATION FIELD (BITS 21-29) IS NOT A LEGAL G-20 OPERATION
2. THE ADDRESS OF THE NEXT COMMAND TO BE EXECUTED IS NOT WITHIN THE LIMITS OF MEMORY.
3. AN OPERAND ILLEGAL TO A G-20 COMMAND IS COMPUTED (THIS FREQUENTLY OCCURS IN MODE 3 ADDRESSING- (SEE SECTION 3) OR WITH A PRECEDING 'OCA').

DIAGNOSTIC PRINTING:

36000	3	305	77	77777	A	+0000000000000000	+00	0000077
1	2	3	4	5	6	7	8	9

1. ADDRESS OF THE COMMAND INSTRUCTION
2. FLAG
3. MODE AND OPCODE
4. INDEX REGISTER
5. ADDRESS
6. 'A' TO INDICATE ADDRESS OPCODE FAULT
7. SIGNED ACCUMULATOR
8. SIGNED EXPONENT
9. CONTENTS OF INDEX REGISTER (IF ONE WAS USED)

EXPONENT OVERFLOW:

IF, DURING COMPUTING, THE EXPONENT OF THE ACCUMULATOR, THE OPERAND ASSEMBLY REGISTER OR THE ARITHMETIC UNIT BECOMES TOO LARGE, AN EXPONENT OVERFLOW INTERRUPT IS GENERATED AND ONE LINE OF DIAGNOSTIC OUTPUT IS PRINTED. THE FORMAT OF THE PRINTING IS THE SAME AS FOR AN ADDRESS OPCODE FAULT WITH THE EXCEPTION THAT FIELD 6 CONTAINS AN 'E' TO INDICATE THAT AN EXPONENT OVERFLOW HAS OCCURRED.

PRINT LINE EXCEEDED:

IF, WHILE STORING CHARACTERS INTO THE PRINT LINE, AN ATTEMPT IS MADE TO STORE A CHARACTER OUTSIDE THE LINE, THE PROGRAM IS HALTED AND THE FOLLOWING IS PRINTED:

<ON THIS LINE IS PRINTED THE CONTENTS OF THE PRINT LINE>
PRINT LINE EXCEEDED

THE FIRST LINE PRINTED IS THE CONTENTS (FIRST 120 COLUMNS) OF THE PRINT LINE AT THE TIME THE ERROR OCCURS. FOLLOWING IS THE MESSAGE 'PRINT LINE EXCEEDED'.

APPENDIX A

PAGE	CONTENT
TH.A.1	G-20 ALPHABET
TH.A.2	G-20 'THAT' OPCODES
TH.A.3	COMMANDS IN NUMERICAL ORDER
TH.A.5	COMMANDS IN ALPHABETICAL ORDER
TH.A.7	SUDOS IN 'THAT'
TH.A.8	G-20 SHIFT MULTIPLIERS
TH.A.9	BRIEF DECIMAL-OCTAL CONVERSION TABLE

G-20 ALPHABET

TH.A.1

SYMBOL	INTERNAL	CARD CODE	SYMBOL	INTERNAL	CARD CODE
SPACE	00	NO PUNCH	0	40	0
A	01	+ 1	1	41	1
B	02	+ 2	2	42	2
C	03	+ 3	3	43	3
D	04	+ 4	4	44	4
E	05	+ 5	5	45	5
F	06	+ 6	6	46	6
G	07	+ 7	7	47	7
H	10	+ 8	8	50	8
I	11	+ 9	9	51	9
J	12	- 1	10	52	0 7 8 NOTE 1
K	13	- 2	.	53	+ 3 8
L	14	- 3	+	54	+
M	15	- 4	-	55	-
N	16	- 5	*	56	- 4 8
O	17	- 6	/	57	0 1
P	20	- 7	=	60	3 8
Q	21	- 8	v	61	+ 7 8 NOTE 1
R	22	- 9	#	62	- 2 8 NOTE 1
S	23	0 2	^	63	+ 6 8 NOTE 1
T	24	0 3	<	64	- 5 8 NOTE 1
U	25	0 4	\$	65	- 3 8
V	26	0 5	>	66	- 6 8 NOTE 1
W	27	0 6	;	67	4 8 NOTE 2
X	30	0 7	(70	0 4 8
Y	31	0 8	[71	0 5 8 NOTE 1
Z	32	0 9]	72	0 6 8 NOTE 1
	33	2 8 NOTE 1)	73	+ 4 8
+	34	6 8 NOTE 1	+	74	7 8 NOTE 1
→	35	- 7 8 NOTE 1	+	75	+ 2 8 NOTE 1
↖	36	+ 5 8 NOTE 1	:	76	0 2 8 NOTE 1
,	37	0 3 8	!	77	5 8 NOTE 2

THE INTERNAL REPRESENTATIONS ABOVE ARE OCTAL INTEGERS.

NOTE 1: MUST BE PUNCHED USING THE MULTIPLE PUNCH BUTTON

NOTE 2: THE KEY MARKED QUOTE ON THE KEYPUNCH ACTUALLY PUNCHES THE SEMI-COLON - THE 4-8 COMBINATION. THE G-20 CHARACTER QUOTE MUST BE MULTI-PUNCHED AS 5-8.

G-20 *THAT* OPCODES

ADDRESS PREPARATION

OCA 000 $X \rightarrow (OA)$
 OCS 020 $\neg X \rightarrow (OA)$
 OAD 040 $(ACC) + X \rightarrow (OA)$
 OSU 060 $(ACC) - X \rightarrow (OA)$
 OSN 120 $\neg(ACC) + X \rightarrow (OA)$
 OAN 100 $\neg(ACC) - X \rightarrow (OA)$
 OAA 140 $|(ACC) + X| \rightarrow (OA)$
 OSA 160 $|(ACC) - X| \rightarrow (OA)$

ADD AND SUBTRACT

CLA 005 $X \rightarrow (ACC)$
 CLS 025 $\neg X \rightarrow (ACC)$
 ADD 045 $(ACC) + X \rightarrow (ACC)$
 SUB 065 $(ACC) - X \rightarrow (ACC)$
 ADN 105 $\neg(ACC) - X \rightarrow (ACC)$
 SUN 125 $\neg(ACC) + X \rightarrow (ACC)$
 ADA 145 $|(ACC) + X| \rightarrow (ACC)$
 SUA 165 $|(ACC) - X| \rightarrow (ACC)$

ARITHMETIC TESTS

FOM 021 $X < 0$
 FOP 001 $X > 0$
 FLD 121 $(ACC) < 0$
 FGD 061 $(ACC) > 0$
 FUD 161 $(ACC) \neq X$
 FSM 101 $(ACC) + X < 0$
 FSN 141 $|(ACC) + X| > 0$
 FSP 041 $(ACC) + X > 0$

MULTIPLY AND DIVIDE

MPY 077 $(ACC) * X \rightarrow (ACC)$
 SUL 075 $(ACC) - X \rightarrow (ACC)$
 DIV 053 $(ACC) / X \rightarrow (ACC)$
 RDV 057 $X / (ACC) \rightarrow (ACC)$

LOGIC OPERATIONS

CAL 015 $X \rightarrow (ACC)$
 CCL 035 $\neg X \rightarrow (ACC)$
 ADL 055 $(ACC) + X \rightarrow (ACC)$
 EXL 115 $(ACC) \wedge X \rightarrow (ACC)$
 ECL 135 $(ACC) \wedge \neg X \rightarrow (ACC)$
 UNL 155 $(ACC) \vee X \rightarrow (ACC)$
 UCL 175 $(ACC) \vee \neg X \rightarrow (ACC)$

LOGIC TESTS

IOZ 011 $X = 0$
 ICZ 031 $\neg X = 0$
 ISN 051 $(ACC) + X \neq 0$
 IUD 071 $(ACC) - X = 0$
 IEZ 111 $(ACC) \wedge X = 0$
 IEC 131 $(ACC) \wedge \neg X = 0$
 IUZ 151 $(ACC) \vee X = 0$
 IUC 171 $(ACC) \vee \neg X = 0$

STORE

STL 173 $(ACC) \rightarrow X$
 STD 153 $(ACC) \rightarrow X, X + 1$
 STS 113 $(ACC) \rightarrow X$
 STI 133 $(ACC) \rightarrow X$
 STZ 073 $0 \rightarrow X$

INDEX REGISTER CODES

LXP 012 $X \rightarrow I$
 LXM 032 $\neg X \rightarrow I$
 ADX 002 $(I) + X \rightarrow I$
 SUX 022 $(I) - X \rightarrow I$
 XPT 016 $X \rightarrow I$ ($\neq 0$)
 XMT 036 $\neg X \rightarrow I$ ($\neq 0$)
 AXT 006 $(I) + X \rightarrow I$ ($\neq 0$)
 SXT 026 $(I) - X \rightarrow I$ ($\neq 0$)

TRANSFER OF CONTROL

TRA 017 $X \rightarrow NC$
 SKP 137 $(NC) + X \rightarrow NC$
 TRM 177 $(NC) \rightarrow X; X + 1 \rightarrow NC$
 REP 013 REPEAT
 XEQ 010 EXECUTE X

MODE	INTERPRETATION
0	$X + (I) + (OA)$
1	$(X) + (I) + (OA)$
2	$(X + (I) + (OA))$
3	$((X) + (I) + (OA))$

FOR ALL TESTS, DO NEXT IF
 CONDITION INDICATED IS TRUE.

THAT ASSEMBLES ALL COMMANDS
 IN MODE 2 EXCEPT:

STI	TRA
STS	TRM
STD	REP
STL	
STZ	

COMMANDS IN NUMERICAL ORDER

000 OCA OPERAND CLEAR ADD	$X \leftarrow (DA)$
001 FOP IF OPERAND PLUS	$X > 0$
002 ADX ADD TO INDEX	$(I) + X \leftarrow I$
005 CLA CLEAR ADD	$X \leftarrow (ACC)$
006 AXT ADD TO INDEX AND TEST	$(I) + X \leftarrow I \quad (\neq 0)$
010 XEQ EXECUTE OPERAND	$X \leftarrow (NC), X+1 \leftarrow (NC)$
011 IOZ IF OPERAND ZERO	$X = 0$
012 LXP LOAD INDEX PLUS	$X \leftarrow I$
013 REP REPEAT	REPEAT
015 CAL CLEAR ADD LOGIC	$X \leftarrow (ACC)$
016 XPT LOAD INDEX PLUS AND TEST	$X \leftarrow I \quad (\neq 0)$
017 TRA TRANSFER	$X \leftarrow (NC)$
020 OCS OPERAND CLEAR SUBTRACT	$-X \leftarrow (DA)$
021 FOM IF OPERAND MINUS	$X < 0$
022 SUX SUBTRACT FROM INDEX	$(I) - X \leftarrow I$
025 CLS CLEAR SUBTRACT	$-X \leftarrow (ACC)$
026 SXT SUBTRACT FROM INDEX AND TEST	$(I) - X \leftarrow I \quad (\neq 0)$
031 ICZ IF COMPLEMENT ZERO	$\neg X = 0$
032 LXM LOAD INDEX MINUS	$-X \leftarrow I$
035 CCL CLEAR ADD COMPLEMENT LOGIC	$\neg X \leftarrow (ACC)$
036 XMT LOAD INDEX MINUS AND TEST	$-X \leftarrow I \quad (\neq 0)$
040 OAD OPERAND ADD	$(ACC) + X \leftarrow (DA)$
041 FSP IF SUM PLUS	$(ACC) + X > 0$
045 ADD ADD	$(ACC) + X \leftarrow (ACC)$
051 ISN IF SUM NON-ZERO	$(ACC) + X \neq 0$
053 DIV DIVIDE	$(ACC) / X \leftarrow (ACC)$
055 ADL ADD LOGIC	$(ACC) + X \leftarrow (ACC)$
057 RDV REVERSE DIVIDE	$X / (ACC) \leftarrow (ACC)$
060 OSU OPERAND SUBTRACT	$(ACC) - X \leftarrow (DA)$
061 FGO IF GREATER THAN OPERAND	$(ACC) > X$
065 SUB SUBTRACT	$(ACC) - X \leftarrow (ACC)$
071 IUO IF UNEQUAL OPERAND	$(ACC) \neq X$
073 STZ STORE ZERO	$0 \leftarrow X$
075 SUL SUBTRACT LOGIC	$(ACC) - X \leftarrow (ACC)$
077 MPY MULTIPLY	$(ACC) * X \leftarrow (ACC)$

100 OAN OPERAND ADD AND NEGATE	$-(ACC) - X \rightarrow (OA)$
101 FSM IF SUM MINUS	$(ACC) + X < 0$
105 ADN ADD AND NEGATE	$-(ACC) - X \rightarrow (ACC)$
111 IEZ IF EXTRACT ZERO	$(ACC) \wedge X = 0$
113 STS STORE SINGLE	$(ACC) \rightarrow X$
115 EXL EXTRACT LOGIC	$(ACC) \wedge X \rightarrow (ACC)$
120 OSN OPERAND SUBTRACT AND NEGATE	$-(ACC) + X \rightarrow (OA)$
121 FLO IF LESS THAN OPERAND	$(ACC) < X$
125 SUN SUBTRACT AND NEGATE	$-(ACC) + X \rightarrow (ACC)$
131 IEC IF EXTRACT COMPLEMENT ZERO	$(ACC) \wedge X = 0$
133 STI STORE INTEGER	$(ACC) \rightarrow X$
135 ECL EXTRACT COMPLEMENT LOGIC	$(ACC) \wedge \neg X \rightarrow (ACC)$
137 SKP SKIP	$(NC) + X \rightarrow NC$
140 OAA OPERAND ADD AND ABSOLUTE	$ (ACC) + X \rightarrow (OA)$
141 FSN IF SUM NON-ZERO	$(ACC) + X \neq 0$
145 ADA ADD AND ABSOLUTE	$ (ACC) + X \rightarrow (ACC)$
151 IUZ IF UNION ZERO	$(ACC) \vee X = 0$
153 STD STORE DOUBLE	$(ACC) \rightarrow X, X + 1$
155 UNL UNITE LOGIC	$(ACC) \vee X \rightarrow (ACC)$
160 OSA OPERAND SUBTRACT AND ABSOLUTE	$ (ACC) - X \rightarrow (OA)$
161 FUD IF UNEQUAL OPERAND	$(ACC) \neq X$
165 SUA SUBTRACT AND ABSOLUTE	$ (ACC) - X \rightarrow (ACC)$
171 IUC IF UNION COMPLEMENT ZERO	$(ACC) \vee X = 0$
173 STL STORE LOGIC	$(ACC) \rightarrow X$
175 UCL UNITE COMPLEMENT LOGIC	$(ACC) \vee \neg X \rightarrow (ACC)$
177 TRM TRANSFER AND MARK	$(NC) \rightarrow X; X + 1 \rightarrow NC$

COMMANDS IN ALPHABETICAL ORDER

145 ADA ADD AND ABSOLUTE	$ (\text{ACC}) + X \rightarrow (\text{ACC})$
045 ADD ADD	$(\text{ACC}) + X \rightarrow (\text{ACC})$
055 ADL ADD LOGIC	$(\text{ACC}) + X \rightarrow (\text{ACC})$
105 ADN ADD AND NEGATE	$-(\text{ACC}) - X \rightarrow (\text{ACC})$
002 ADX ADD TO INDEX	$(I) + X \rightarrow I$
006 AXT ADD TO INDEX AND TEST	$(I) + X \rightarrow I \quad (\neq 0)$
015 CAL CLEAR ADD LOGIC	$X \rightarrow (\text{ACC})$
005 CLA CLEAR ADD	$X \rightarrow (\text{ACC})$
035 CCL CLEAR ADD COMPLEMENT LOGIC	$\neg X \rightarrow (\text{ACC})$
025 CLS CLEAR SUBTRACT	$-X \rightarrow (\text{ACC})$
053 DIV DIVIDE	$(\text{ACC}) / X \rightarrow (\text{ACC})$
135 ECL EXTRACT COMPLEMENT LOGIC	$(\text{ACC}) \wedge \neg X \rightarrow (\text{ACC})$
115 EXL EXTRACT LOGIC	$(\text{ACC}) \wedge X \rightarrow (\text{ACC})$
061 FGD IF GREATER THAN OPERAND	$(\text{ACC}) > X$
121 FLD IF LESS THAN OPERAND	$(\text{ACC}) < X$
021 FOM IF OPERAND MINUS	$X < 0$
001 FOP IF OPERAND PLUS	$X > 0$
101 FSM IF SUM MINUS	$(\text{ACC}) + X < 0$
141 FSN IF SUM NON-ZERO	$(\text{ACC}) + X \neq 0$
041 FSP IF SUM PLUS	$(\text{ACC}) + X > 0$
161 FUD IF UNEQUAL OPERAND	$(\text{ACC}) \neq X$
031 ICZ IF COMPLEMENT ZERO	$\neg X = 0$
131 IEC IF EXTRACT COMPLEMENT ZERO	$(\text{ACC}) \wedge X = 0$
111 IEZ IF EXTRACT ZERO	$(\text{ACC}) \wedge X = 0$
011 IOZ IF OPERAND ZERO	$X = 0$
051 ISN IF SUM NON-ZERO	$(\text{ACC}) + X \neq 0$
171 IUC IF UNION COMPLEMENT ZERO	$(\text{ACC}) \vee X = 0$
071 IUD IF UNEQUAL OPERAND	$(\text{ACC}) \neq X$
151 IUZ IF UNION ZERO	$(\text{ACC}) \vee X = 0$
032 LXM LOAD INDEX MINUS	$-X \rightarrow I$
012 LXP LOAD INDEX PLUS	$X \rightarrow I$
077 MPY MULTIPLY	$(\text{ACC}) * X \rightarrow (\text{ACC})$
140 OAA OPERAND ADD AND ABSOLUTE	$ (\text{ACC}) + X \rightarrow (\text{OA})$
040 OAD OPERAND ADD	$(\text{ACC}) + X \rightarrow (\text{OA})$
100 OAN OPERAND ADD AND NEGATE	$-(\text{ACC}) - X \rightarrow (\text{OA})$

000 OCA OPERAND CLEAR ADD
 020 OCS OPERAND CLEAR SUBTRACT
 160 OSA OPERAND SUBTRACT AND ABSOLUTE
 120 OSN OPERAND SUBTRACT AND NEGATE
 060 OSU OPERAND SUBTRACT
 057 RDV REVERSE DIVIDE
 013 REP REPEAT
 137 SKP SKIP
 153 STD STORE DOUBLE
 133 STI STORE INTEGER
 173 STL STORE LOGIC
 113 STS STORE SINGLE
 073 STZ STORE ZERO
 165 SUA SUBTRACT AND ABSOLUTE
 165 SUB SUBTRACT
 075 SUL SUBTRACT LOGIC
 125 SUN SUBTRACT AND NEGATE
 022 SUX SUBTRACT FROM INDEX
 026 SXT SUBTRACT FROM INDEX AND TEST
 017 TRA TRANSFER
 177 TRM TRANSFER AND MARK
 175 UCL UNITE COMPLEMENT LOGIC
 155 UNL UNITE LOGIC
 010 XEQ EXECUTE OPERAND
 016 XPT LOAD INDEX PLUS AND TEST
 036 XMT LOAD INDEX MINUS AND TEST

$X \rightarrow (DA)$
 $- X \rightarrow (DA)$
 $| (ACC) - X | \rightarrow (DA)$
 $- (ACC) + X \rightarrow (DA)$
 $(ACC) - X \rightarrow (DA)$
 $X / (ACC) \rightarrow (ACC)$
 REPEAT
 $(NC) + X \rightarrow NC$
 $(ACC) \rightarrow X, X + 1$
 $(ACC) \rightarrow X$
 $(ACC) \rightarrow X$
 $(ACC) \rightarrow X$
 $0 \rightarrow X$
 $| (ACC) - X | \rightarrow (ACC)$
 $(ACC) - X \rightarrow (ACC)$
 $(ACC) - X \rightarrow (ACC)$
 $- (ACC) + X \rightarrow (ACC)$
 $(I) - X \rightarrow I$
 $(I) - X \rightarrow I \quad (\neq 0)$
 $X \rightarrow (NC)$
 $(NC) \rightarrow X; X + 1 \rightarrow NC$
 $(ACC) \vee \neg X \rightarrow (ACC)$
 $(ACC) \vee X \rightarrow (ACC)$
 EXECUTE X AS COMMAND
 $X \rightarrow I \quad (\neq 0)$
 $- X \rightarrow I \quad (\neq 0)$

SUDOS IN 'THAT'

ADC	ADDRESS CONSTANT
ALF	ALPHANUMERIC INFORMATION
CHK	CHECK
COM	COMMENT
CPY	COPY
DBG	DEBUG
DEC	DECIMAL LISTING
DEF	DEFINE
DMP	DUMP
ENT	ENTRY
FLG	FLAG
FPC	FULL PRECISION CONSTANT
HPC	HALF PRECISION CONSTANT
LBL	LABEL
LIN	LINE
LWD	LOGIC WORD
MTT	MARK TRANSFER TO
NAM	NAME
OCT	OCTAL LISTING
OPM	OPERATOR MESSAGE
OUT	OUT
PAG	PAGE
PBC	PUNCH BINARY CARDS
PRT	PRINT
RBC	READ BINARY CARDS
REL	RELEASE
RET	RETURN
RGN	PRINT REGIONAL SYMBOL
SXX	SET STORAGE EXTRACTOR
TOP	TYPE OR PRINT
WRD	WORD

G-20 SHIFT MULTIPLIERS

LEFT SHIFT	NUMBER	RIGHT SHIFT
1	0	000 00 00001
2	1	101 00 00004
4	2	101 00 00002
10	3	101 00 00001
20	4	102 00 00004
40	5	102 00 00002
100	6	102 00 00001
200	7	103 00 00004
400	8	103 00 00002
1000	9	103 00 00001
2000	10	104 00 00004
4000	11	104 00 00002
10000	12	104 00 00001
20000	13	105 00 00004
40000	14	105 00 00002
05 00 00001	15	105 00 00001
05 00 00002	16	106 00 00004
05 00 00004	17	106 00 00002
06 00 00001	18	106 00 00001
06 00 00002	19	107 00 00004
06 00 00004	20	107 00 00002
07 00 00001	21	107 00 00001
07 00 00002	22	110 00 00004
07 00 00004	23	110 00 00002
10 00 00001	24	110 00 00001
10 00 00002	25	111 00 00004
10 00 00004	26	111 00 00002
11 00 00001	27	111 00 00001
11 00 00002	28	112 00 00004
11 00 00004	29	112 00 00002
12 00 00001	30	112 00 00001
12 00 00002	31	113 00 00004

BRIEF DECIMAL-OCTAL CONVERSION TABLE

DECIMAL	OCTAL	OCTAL	DECIMAL
10	12	10	8
20	24	20	16
30	36	30	24
40	50	40	32
50	62	50	40
60	74	60	48
70	106	70	56
80	120		
90	132	100	64
		200	128
100	144	300	192
200	310	400	256
300	454	500	320
400	620	600	384
500	764	700	448
600	1 130		
700	1 274	1 000	512
800	1 440	2 000	1 024
900	1 604	3 000	1 536
		4 000	2 048
1 000	1 750	5 000	2 560
2 000	3 720	6 000	3 072
3 000	5 670	7 000	3 584
4 000	7 640		
5 000	11 610	10 000	4 096
6 000	13 560	20 000	8 192
7 000	15 530	30 000	12 288
8 000	17 500	40 000	16 384
9 000	21 450	50 000	20 480
		60 000	24 576
10 000	23 420	70 000	28 672
20 000	47 040		
30 000	72 460	100 000	32 768
40 000	116 100	200 000	65 536
50 000	141 520	300 000	98 304
60 000	165 140	400 000	131 072
70 000	210 560	500 000	163 840
80 000	234 200	600 000	196 608
90 000	257 620	700 000	229 376
100 000	303 240	1 000 000	262 134