

# **ASSEMBLY LANGUAGE**

## ASSEMBLY LANGUAGE

Symbolic Coding	.....
Instruction Form	.....
Types of Symbols	.....
Instruction Content	.....
Operation Codes	.....
Class 0, Tests and Transfers	
Class 1, Arithmetic	
Class 2, Fetch, Store, Tags	
Class 4, B-Registers, Lights, Special Registers, Shifts	
Class 5, Logic and Fast Registers	
Class 6, Input-Output	
Class 7, Analog Input, Shifts, Delays	
Summary of Operation Codes	
Pseudo-Orders	.....
ORG and END	
EQU	
BSS and BES	
BCD, FLX, REM	
DEC and OCT	
REF	
Macro-Orders	.....
Application	
Definition	
Call	
Examples	
Assembly Procedure	.....
Coding Examples	.....

## SYMBOLIC CODING

The absolute machine language of the Rice Computer is described in detail in the Rice Computer Manual. In practice, programs are not written in the absolute language of the computer but in a symbolic language. A language which provides symbolic notation for instructions, or commands, that correspond one-for-one with absolute machine instructions is called an assembly language. The program which translates assembly language into machine language is called an assembly program.

Use of the assembly language for the Rice Computer depends on a knowledge of the absolute machine instruction format, a familiarity with the registers of the computer, and a general acquaintance with the instruction repertoire -- all explained in the Rice Computer Manual. Two forms of the Rice Computer assembly language are available:

AP1, for independent use

AP2, for use within Genie programs

The corresponding assembly programs have the same names:

AP1, an independent assembly program

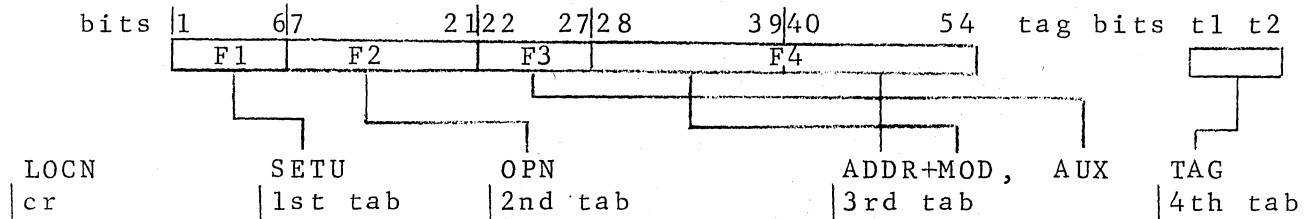
AP2, a subset of the Genie compiler

The two assembly languages are very similar. The major distinction concerns octal and decimal numerals. In AP1, all numeric constants are assumed to be octal unless immediately preceded by the special symbol "d", meaning decimal. In AP2, all numeric constants are assumed to be decimal, except when octal form is indicated by a plus sign immediately preceding the octal number.

In the following discussions, M stands for the final number formed in the last 15 bits of I (the instruction register) after all specified indirect addressing and B-modification has taken place; and if Q is any machine location, then (Q) stands for the contents of location Q.

## INSTRUCTION FORM

The general form of an AP1 or AP2 instruction and its correspondence to a machine-language instruction as explained in the Rice Computer Manual is



Here "cr" denotes "carriage return", and "tab" denotes "tabulate" on the flexowriter used for preparation of input to the assembly programs.

LOCN gives the symbolic label (if any) on the instruction.

SETU corresponds to Field 1: bring a "fast" register to U; then inflect (U).

OPN corresponds to a Field 2 operation chosen from one of seven classes.

AUX corresponds to Field 3: alter a B-register, send (U) or (R) to a "fast" register, send the M portion of I to a B-register, or clear R.

ADDR+MOD corresponds to Field 4: compute the final address M, sending M to the last 15 bits of I; load S with M or (M); then inflect (S).

All fields may be symbolically coded. All fields but MOD and TAG may be coded numerically.

If no TAG is to be specified, the 4th tab may be omitted. If no AUX operation is to be specified, the preceding comma may be omitted.

## TYPES OF SYMBOLS

Precise definitions of the allowed symbols are as follows:

Register names. The following symbols are used as names of "fast" registers:

A-series            Z, U, R, S, T4, T5, T6, T7

B-series            CC, B1, B2, B3, B4, B5, B6, PF

These may appear in SETU, ADDR+MOD, and AUX fields. The symbol I may be used in SETU and AUX. The special register names may be used in ADDR; these are

SL            sense lights

IL            indicator lights

ML            mode lights

TL            trapping lights

P2            second pathfinder

X            increment register

TT            "to-tape" register

FT            "from-tape" register

These symbols may be used only as register names.

Special characters. \*, a(AP1) or #(AP2), d(AP1), +, -, |, -, (, ), "tab", "cr", and , (comma).

Operation codes. These include the mnemonic operation codes in the assembly vocabulary, pseudo-operation codes (AP1 only), macro-operations (AP1 only), and general symbols defined by the user as operation codes with a LET (in Genie for AP2) or an EQU (in AP1). All of these areas are covered in later discussions.

General names. In AP2, a private name may be

a single lower case Roman letter

or an upper case Roman letter, followed by upper case Roman letters, followed by lower case Roman letters, followed by numerals.

In AP1, a private name may be

an upper case Roman letter, followed by upper case Roman letters, followed by numerals.

Spaces may not appear in names. Any number of characters may form

## TYPES OF SYMBOLS

2

a name; AP2 will retain the first four if lower case Roman letters are used, the first five otherwise; AP1 will retain the first six. The following are general names in AP1 and AP2: B, M3, COMM, ZETA2. The following are general names in AP2, but not in AP1: b, Comm, Zeta2. General names may appear only in the LOCN and ADDR fields.

## INSTRUCTION CONTENT

Each field of the symbolic instruction has a well-defined form. If this form is not recognized by the assembly system, a message is printed during assembly. The acceptable contents of each field are as follows:

LOCN. This field may be blank or absolute or symbolic. Absolute LOCN fields are permitted only when an APl program is being assembled in absolute form (see the ORG pseudo-order discussion). A symbolic LOCN field may contain any general name. A name may not appear in LOCN more than once in any one program.

SETU. This field may be blank, absolute, or F, where F is an A- or B-series register name or "I", or any of the forms -F, |F|, or -|F|. If SETU is blank, "U" is understood and the octal equivalent 01 is inserted into the machine instruction. I sets U to the integer +1; -I sets U to the integer -1. Note that Z sets U to all zeroes; -Z sets U exponent to zero and U mantissa to minus zero, or all ones.

Examples: B1    |T4|    -PF    -|R|    Z    -I

If the T-flag is on for register Ti (i=4,5,6,7), indirect addressing through Ti will occur when Ti is addressed in the SETU field. To denote this mode of addressing the \* may be used before the register name:

\*Ti    -\*Ti    |\*Ti|    -|\*Ti|

This is a symbolic convenience only, and these will be translated as:

Ti    -Ti    |Ti|    -|Ti|

OPN. This field may be absolute or an operation code. In the case of conditional transfers, a symbolic operation has the form IF(CCC)TTT where CCC represents test conditions and TTT is a mnemonic for a transfer order. Other symbolic operation codes consist of

one or more 3-letter mnemonics. Special symbols such as  $\rightarrow$ ,  $+$ ,  $-$ ,  $"$ ,  $'$ , and  $+i$  (where  $i$  is an octal integer) are sometimes permitted (see the section on operation codes).

AUX. This field may be blank, absolute, or one of the forms  $U \rightarrow F$ ,  $R \rightarrow F$ ,  $I \rightarrow B_i$ ,  $B_i + 1$ ,  $B_i - 1$ , or  $B_i + X$ .  $B_i$  stands for one of the B-series register names;  $F$  is any A- or B-series register name;  $I$  refers to the last 15 bits of the instruction register; and  $X$  is the increment register. As a special case,  $R \rightarrow Z$  causes  $R$  to be cleared to zero.

Example:  $U \rightarrow T4$      $R \rightarrow PF$      $I \rightarrow B1$      $B2 + 1$      $B3 - 1$      $B4 + X$

If the T-flag is on for register  $T_i$  ( $i=4,5,6,7$ ), indirect addressing through  $T_i$  will occur when  $T_i$  is addressed in the AUX field. To denote this mode of addressing the  $*$  may be used before the register name:

$U \rightarrow *T_i$      $R \rightarrow *T_i$

This is a symbolic convenience only, and these will be translated exactly as:

$U \rightarrow T_i$      $R \rightarrow T_i$

ADDR+MOD. ADDR may be blank or absolute or symbolic, or the ADDR+MOD field may consist of an octal or decimal number to be used as an operand. MOD is either blank or one or more of the B-series register names, connected to ADDR by  $+$  signs. Special inflections control the IM and IA bits as follows: IM bit 1 is set to 1 (to load  $S$  with  $M$  instead of  $(M)$ ) whenever the symbol "a" (AP1) or "#" (AP2) appears, or whenever certain OPN mnemonics are used (see the section on operation codes). IM bits 2 (absolute value) and 3 (minus) are controlled by the special forms  $-Q$ ,  $|Q|$ , and  $-|Q|$ , where  $Q$  is an allowed ADDR+MOD symbol. The IA (indirect addressing) bit is set to 1 whenever the symbol "\*" appears in this field.

If ADDR is symbolic, any A-series register name, any special register name, or any general name is acceptable. A general name may be followed by a relative part consisting of an integer preceded



by a + or - sign.

If ADDR is absolute, any octal integer of not more than 5 digits, or any decimal integer of absolute value not larger than 32,767, is permissible. Any octal or decimal integer above these limits or any floating point decimal number is treated as the name of a location containing that number; storage space is reserved for it at the end of the program. In this case, no MODs are allowed, and only the absolute value and - inflections are meaningful.

All characters appearing within parentheses in this field are ignored, so that an address field which is modified by the program may be conveniently noted. For example, (FWA)+B1+B2 is treated as Z+B1+B2. If a symbol appears in ADDR but never in LOCN, a blank location will be reserved at the end of the program. ADDR+MOD should not be blank; the Z character may always be used to produce a zero field.

Examples of equivalent AP1 and AP2 ADDR+MOD fields are:

AP1	AP2
COMM+10 or COMM+d8	COMM+8 or COMM++10
- A+B1-d12  or - A+B1-14	- A+B1-12  or - A+B1-+14
a*ZETA	#*ZETA
d48	48
-ad122+B1	-#122+B1
B4+B5	B4+B5
00500	+00500
d2.009027	2.009027
777700000	+777700000
30	24

The only field which may be continued onto another line is ADDR+MOD, AUX by punching a "cr" followed immediately by three "tab" characters, so that continuation lines will follow under ADDR+MOD, AUX.

TAG. This field may be blank or symbolic. If no tag is desired, the 4th tab punch may be omitted. If a tag is desired, the TAG field must contain one of the mnemonics TG1, TG2, or TG3. The corresponding tag will be placed on the assembled instruction, printed on the octal listing, and punched with the instruction in checksum format.

## OPERATION CODES

The most common Field 2 operations have been given names in the vocabulary of AP1 and AP2 for convenience in coding. All Field 2 operations are fully explained in the machine manual. The mnemonics defined in this section are summarized in a chart at the end of the section. These operation code symbols may not be used for any other purpose. Other Field 2 operations may be given general names by use of LET (in Genie for AP2) or EQU (AP1), and such symbols are then treated as operation codes throughout the program in which they have been defined.

• Class 0, Tests and Transfers

In the list below, the symbols are followed by their octal equivalents and a brief explanation of their meanings; the indication "a,#" means that the operation symbol automatically causes IM bit 1 to be set to 1 (to load S with M instead of (M)), since the operation indicated deals with M rather than with (S).

The four unconditional transfers are represented by:

octal codes		
a,#	HTR 00000	Halt and transfer. Halt, setting CC to M when CONTINUE is pressed.
a,#	TRA 01000	Transfer. Set CC to M.
	SKP 02000	Skip. Subtract (S) from (U); then increment CC by 1, skipping the next order.
	JMP 03000	Jump. Subtract (S) from (U); then increment CC by (X), the increment register.

Conditional transfers have the form IF(CCC)TTT where TTT is one of the above transfer mnemonics, and CCC represent one, two, or three test conditions joined by + or x signs. Use of the + sign indicates that the specified transfer is to occur if any of the conditions listed is satisfied; use of the x sign indicates that the specified transfer occurs only when all of the conditions listed are satisfied simultaneously. A single order may not contain both + and x signs. One condition from each of the first three groups may be specified; or a Group IV mnemonic may be combined with a Group III test as noted. If a TRA or HTR is used, the specified test is made on (U). If a SKP or JMP is used, the specified test is normally performed on (U)-(S). The exceptions to this rule are noted below Group II.

Group I

	octal code	
PSN	00100	Positive sign. Is the sign bit of U equal to 0?
MOV	00200 *	Mantissa overflow. Is Indicator Light #4 on?
EOV	00300 *	Exponent overflow. Is Indicator Light #5 on?
NSN	00500	Negative sign. Is the sign bit of U equal to 1?
NMO	00600 *	No mantissa overflow. Is Indicator Light #4 off?
NEO	00700 *	No exponent overflow. Is Indicator Light #5 off?

\*Note that indicator lights are turned off when tested.

Group II

	octal code	
ZER	00010	Zero. Is (U) mantissa all 1's or all 0's?
EVN	00020	Even. Is bit 54 of U equal to zero?
a, # SLN	00030 *	Sense light on. Are all the sense lights corresponding to 1's in M on?
NUL	00040 **	Null. Are all 54 bits of U zero?
NZE	00050	Non-zero. Is (U) mantissa different from zero?
ODD	00060	Odd. Is bit 54 of U equal to 1?
a, # SLF	00070 *	Sense light off. Are all the sense lights corresponding to 1's in M off?

\*Note that sense lights are not altered when tested. SLN and SLF tests are meaningful only with SKP or JMP orders, and in these cases no subtraction takes place.

\*\*If the NUL test is used with a SKP or JMP order, a logical comparison is made as follows: wherever a bit of R is equal to zero, the

bits in corresponding positions of U and S are compared. If (U) is identical with (S) in each of these positions, the resulting (U) is null and the NUL portion of the test is satisfied. If the NUL comparison is not satisfied, the resulting (U) is meaningless.

Group III

	octal code	
TG1	00001*	Tag 1. Is Indicator Light #1 on?
TG2	00002*	Tag 2. Is Indicator Light #2 on?
TG3	00003*	Tag 3. Is Indicator Light #3 on?
NTG	00004*	No tag. Are Indicator Lights #1, #2, #3 all off?
NT1	00005*	No tag 1. Is Indicator Light #1 off?
NT2	00006*	No tag 2. Is Indicator Light #2 off?
NT3	00007*	No tag 3. Is Indicator Light #3 off?

\*Note that indicator lights are turned off when tested.

Group IV

	octal code	
POS	00110	Positive <u>or</u> zero. Is (U) mantissa greater than or equal to zero?
NEG	00510	Negative <u>or</u> zero. Is (U) mantissa less than or equal to zero?

A + sign must be used when combining either of these mnemonics with a Group III test.

	octal code	
PNZ	04150	Positive <u>and</u> non-zero. Is (U) mantissa strictly greater than zero?
NNZ	04550	Negative <u>and</u> non-zero. Is (U) mantissa strictly less than zero?

A x sign must be used when combining either of these mnemonics with a Group III test.

• Class 1, Arithmetic

In the list below, the symbols are followed by their octal equivalents and a brief explanation of their meanings.

Any Class 1 mnemonic may be followed by  $\rightarrow$  or +1, to cause storing of the final (U) in the location addressed by M; by +2, storing (U) at location (B6); or by +3, storing (U) at location  $M+(B6)$ . Octal codes may be joined by a '+' to Class 1 mnemonics for various special operations. If n is such an octal code, the combination appears as

mnemonic +n                      in AP1

mnemonic ++n                      in AP2

Any floating point mnemonic may be followed by +1j (j=0, 1, 2, or 3), causing the last bit of (U) to be set to 1 (rounded) after the operation but before storing. After floating point mnemonics +4j suppresses normalization of the result, +5j rounds and suppresses normalization. Other options are given in the machine manual.

The Class 1 mnemonics are as follows:

Fixed point

	octal code	
ADD	10000	Add. $(U)+(S) \rightarrow U$ .
SUB	10100	Subtract. $(U)-(S) \rightarrow U$ .
BUS	14100	Reverse subtract. $(S)-(U) \rightarrow U$ .
MPY	10200	Multiply. $(U) \times (S) \rightarrow U, R$ (double length).
IMP	10220	Integer multiply. $(U) \times (S) \rightarrow U$ .
DIV	10300	Divide. Double length $(U, R) \div (S) \rightarrow U$ , $2^{47} \times$ remainder $\rightarrow R$ .
VID	16300	Reverse divide. $(S) \div (U) \rightarrow U$ , $2^{47} \times$ remainder $\rightarrow R$ .
IDV	13300	Integer divide. $(U) \div (S) \rightarrow U$ , remainder $\rightarrow R$ .
VDI	17300	Reverse integer divide. $(S) \div (U) \rightarrow U$ , remainder $\rightarrow R$ .

Floating Point

	octal code	
FAD	10400	Floating add. $(U)+(S) \rightarrow U$ .
FSB	10500	Floating subtract. $(U)-(S) \rightarrow U$ .
BSF	14500	Reverse floating subtract. $(S)-(U) \rightarrow U$ .
FMP	10600	Floating multiply. $(U) \times (S) \rightarrow U, R$ (double length).
FDV	10700	Floating divide. Double length $(U, R) \div (S) \rightarrow U, 2^{47} \times \text{remainder} \rightarrow R$ .
VDF	16700	Reverse floating divide. $(S) \div (U) \rightarrow U,$ $2^{47} \times \text{remainder} \rightarrow R$ .



⊙ Class 2, Fetch, Store, Tags

In the list below, the symbols are followed by their octal equivalents and a brief explanation of their meanings; the indication "a, #" means that the operation symbol automatically causes IM bit 1 to be set to 1 (to load S with M instead of (M)), since the operation indicated deals with M rather than with (S).

Any Group I or Group II mnemonic may be followed by a comma and any Group III mnemonic. In addition, any Group I or Group III mnemonic may be followed by  $\rightarrow$  or +1, storing (U) with (ATR) at location M; or by +2, storing (U) with (ATR) at location (B6); or any Group I, II, or III mnemonic may be followed by +3, storing (U) with (ATR) at location  $M+(B6)$ . Note that all Group I and Group II mnemonics clear (ATR) unless followed by a Group III mnemonic.

The Class 2 mnemonics are as follows:

Group I

	octal code	
CLA	21700	Clear and add. Bring (S) to U.
BEU	21000*	Bring exponent to U. Exponent portion of (S) replaces exponent portion of (U).
BMU	20700*	Bring mantissa to U. Mantissa portion of (S) replaces mantissa portion of (U).
BLU	21400*	Bring left half to U. Left half of (S) replaces left half of (U).
BRU	20300*	Bring right half to U. Right half of (S) replaces right half of (U).
BIU	20200*	Bring inflections to U. Inflection portion of (S) replaces inflection portion of (U).
BAU	20100*	Bring address to U. Address portion of (S) replaces address portion of (U).
BNA	21600*	Bring all except address to U. Inflection and left portions of (S) replace inflection and left portions of (U).

\*The "bring" mnemonics may be joined by commas to fetch more than one portion of a word.

Group II

	octal code	
RPE	20701*	Replace exponent. Exponent portion of (U) replaces exponent portion of word at location M.
RPM	21001*	Replace mantissa. Mantissa portion of (U) replaces mantissa portion of word at location M.
RPL	20301*	Replace left half. Left half of (U) replaces left half of word at location M.
RPR	21401*	Replace right half. Right half of (U) replaces right half of word at location M.
RPA	21601*	Replace address. Address portion of (U) replaces address portion of word at location M.
RPI	21501*	Replace inflections. Inflection portion of (U) replaces inflection portion of word at location M.
a,#	STO 20001	Store. Store (U) at location M.

\*The "replace" mnemonics may not be combined with each other.

Group III

	octal code	
ST1	20010	Set Tag 1. Set ATR to 1.
ST2	20020	Set Tag 2. Set ATR to 2.
ST3	20030	Set Tag 3. Set ATR to 3.
WTG	20040	With Tag. Do not change ATR.

Group IV

	octal code	
NOP	30000	No operation. Do not alter (U) or (ATR).
FST	20041	Fetch and store. Bring contents of location M to S; then store (U) with (ATR) at location M.
RWT	21641	Replace address, with tag. Address portion of (U) replaces address portion of word at location M, without changing the tag on the word at location M.

Double Option

Any Class 2 operation applied to U with original F4 address N may also be applied to R with original F4 address N+1 by use of the mnemonic:

octal code

DBL 20004

Double. After operating on U with original F4 address N, apply same operation to R with original F4 address N+1.

Examples:

BAU,DBL DATA

loads the address portion of U from the location DATA and loads the address portion of R from the location DATA +1.

STO,DBL \*ANS

stores (U) through the codeword at location ANS and stores (R) through the codeword at location ANS +1.

Use of the +2 store option with DBL stores (U) with (ATR) at location (B6), stores (R) with (ATR) at location (B6+1), and increments (B6) by 1. The +3 store option with DBL uses (B6) for both stores and does not increment (B6).

After a double operation, the M portion of (I) contains the final address used with R.

• Class 4, B-Registers, Lights, Special Registers, Shifts

In the list below, the symbols are followed by their octal equivalents and a brief explanation of their meanings; the indication "a,#" means that the operation symbol automatically causes IM bit 1 to be set to 1 (to load S with M instead of (M)), since the operation indicated deals with M rather than with (S).

The Class 4 mnemonics are as follows:

B-registers

	octal code	
a,# TSR 40000		Transfer to subroutine. Set PF to (CC); then set CC to M.
a,# SBi 4000i		Set Bi. Set Bi to M, for i=1, 2, ..., 6.
a,# SPF 40007		Set PF. Set PF to M.
a,# ACC 41000		Add to CC. (CC)+M→CC.
a,# ABi 4100i		Add to Bi. (Bi)+M→Bi, for i=1, 2, ..., 6.
a,# APF 41007		Add to PF. (PF)+M→PF.
ERM 00020		Enter repeat mode. Turn on mode light #2.

The ERM mnemonic is meaningful only when joined by a comma to one of the above Class 4 mnemonics.

Lights

	octal code	
a,# SLN 42000		Sense lights on. Turn on sense lights corresponding to 1's in M.
a,# ILN 42001		Indicator lights on. Turn on indicator lights corresponding to 1's in M.
a,# MLN 42002		Mode lights on. Turn on mode lights corresponding to 1's in M.
a,# TLN 42003		Trap lights on. Turn on trapping lights corresponding to 1's in M.
a,# SLF 42004		Sense lights off. Turn off sense lights corresponding to 1's in M.
a,# ILF 42005		Indicator lights off. Turn off indicator lights corresponding to 1's in M.

		octal code	
a,#	MLF	42006	Mode lights off. Turn off mode lights corresponding to 1's in M.
a,#	TLF	42007	Trap lights off. Turn off trapping lights corresponding to 1's in M.

Note that lights corresponding to 0's in M are not affected by the above orders.

#### Special registers

		octal code	
a,#	STX	43005	Set X. Set the increment register to M.
a,#	STT	43006	Set TT. Set the to-tape register to M.
a,#	SFT	43007	Set FT. Set the from-tape register to M.

#### Shifts

		octal code	
a,#	DMR	44000	Double mantissa right. Arithmetic right shift of (U,R) mantissa M places.
a,#	DML	44010	Double mantissa left. Arithmetic left shift of (U,R) mantissa M places.
a,#	LUR	45010	Logical U right. Shift (U) right M places, shifting zeros into left end of U.
a,#	LUL	45020	Logical U left. Shift (U) left M places, shifting zeros into right end of U.
a,#	LRR	45001	Logical R right. Shift (R) right M places, shifting zeros into left end of R.
a,#	LRL	45002	Logical R left. Shift (R) left M places, shifting zeros into right end of R.
a,#	LRS	45015	Long right shift. Shift (U,R) right M places, shifting (U) into R and zeros into left end of U.
a,#	LLS	45062	Long left shift. Shift (U,R) left M places, shifting (R) into U and zeros into right end of R.

## OPERATION CODES

12

	octal code	
a,# CRR	45055	Circle right. Shift (U,R) right M places, shifting (U) into R and right end of (R) into left end of U.
a,# CRL	45066	Circle left. Shift (U,R) left M places, shifting (R) into U and left end of (U) into right end of R.
a,# BCT	46000	Bit count. Clear U; shift R right M places; add each 1 which spills from R one at a time into U.

T-flags

TFU 47000

*No longer operational*

T-flags and ITR to U. Clear U, then bring two ITR and four T-flag bits to U: ITR in octal (0,1,2, or 3) → bits 49 and 50, TF4→bit 51, TF5→bit 52, TF6→bit 53, TF7→bit 54.

• Class 5, Logic and Fast Registers

In the list below, the symbols are followed by their octal equivalents and a brief explanation of their meanings.

Any Class 5 mnemonic may be followed by  $\rightarrow$  or  $+1$ , to cause storing of the final (U) at location M; by  $+2$ , storing (U) at location (B6); or by  $+3$ , storing (U) at location  $M+(B6)$ . In addition, any Class 5 mnemonic may be preceded by a  $-$  sign, causing the final result in U to be complemented (before storing). The Class 5 mnemonics are as follows:

	octal code	
CPL	50100	Complement. Change all 1's in U to 0's and all 0's to 1's.
XUR	54000	Exchange (U) and (R). (U) $\rightarrow$ R as (R) $\rightarrow$ U.
LDU	50410	Load U. (S) $\rightarrow$ U.
LDR	50400	Load R. (S) $\rightarrow$ R without disturbing (U).
LTi	504i0	Load Ti. (S) $\rightarrow$ Ti without disturbing (U) or (R), for $i=4, 5, 6, 7$ .
STF	50540	Set T-flag. Turn on flag bit for the T-register being loaded to cause indirect addressing in F1 and F3. Meaningful only if adjoined to LTi by comma.
SUR	53000	Shuffle S, U, and R. (U) $\rightarrow$ R then (S) $\rightarrow$ U.
ORU	50010	Or to U. Logical or for each bit position: (U)=0 and (S)=0 results in (U)=0; otherwise, (U)=1 as result.
AND	50314	And. Logical and for each bit position: (U)=1 and (S)=1 results in (U)=1; otherwise, (U)=0 as result.
XTR	50020	Extract. For each bit position: (S) $\rightarrow$ U if (R)=1, (U) unchanged if (R)=0.
SYD	53220	Symmetric difference. For each bit position: (U)=(S) results in (U)=0; (U) $\neq$ (S) results in (U)=1.
SYS	53120	Symmetric sum. For each bit position: (U)=(S) results in (U)=1; (U) $\neq$ (S) results in (U)=0.

lv B5 except when T7=1  
 $\neq \rightarrow U$  when T7=1

• Class 6, Input-Output

In the list below, the symbols are followed by their octal equivalents and a brief explanation of their meanings; the indication "a, #" means that the operation symbol automatically causes IM bit 1 to be set to 1 (to load S with M instead of (M)), since the operation indicated deals with M rather than with (S).

For detailed explanations of reading, printing, punching, plotting, and magnetic tape operation, see the Rice Computer Manual.

The Class 6 mnemonics are as follows:

Paper tape

		octal code	
a, #	RTR	60000 *	Read triads. Read 1 to 18 triads from paper tape into U.
a, #	RHX	60100 *	Read hexads. Read 1 to 9 hexads from paper tape into U.
	PHX	60400	Punch hexads. Punch 1 to 9 hexads from (S) onto paper tape.
	PH7	60500	Punch hexads with 7th hole. Punch 1 to 9 hexads, each with a 7th hole, from (S) onto paper tape.
	PTR	60600	Punch triads. Punch 1 to 18 triads from (S) onto paper tape.

\*Either "Read" mnemonic may be followed by  $\rightarrow$  or +1, storing (U) at location M; by +2, storing (U) at location (B6); by +3, storing (U) at location M+(B6); by +40 to turn on IL4 (mantissa overflow) if there is no tape in the reader.

Console typewriter

		octal code	
	TYP	60700	Type. Type (S) as 18 octal digits on console typewriter.

Printer

		octal code	
a, #	PRN	61110	Print numeric. Print, using first 32 characters of print wheel, from print matrix beginning at location M; space one line after printing.



# OPERATION CODES

15

		octal code	
a, #	PRA	61210	Print alphanumeric. Print as above, using all characters.
a, #	PRO	61310	Print octal. Print as above, using characters 0-7 only.
	SPA	61010	Space. Advance printer paper one line.
	SP2	61020	Space, format 2. Advance printer paper to next 1/22 page mark.
	SP3	61030	Space, format 3. Advance printer paper to next 1/11 page mark.
	SP4	61040	Space, format 4. Advance printer paper to next 1/6 page mark.
	SP5	61050	Space, format 5. Advance printer paper to next 1/3 page mark.
	SP6	61060	Space, format 6. Advance printer paper to next 1/2 page mark.
	PAG	61070	Page restore. Advance printer paper to next new page.
	DLY	61000	Printer delay. n successive executions of DLY will delay the machine for at least n-1 tenths of a second and not more than n tenths of a second.

*62000 a2 — disc operation (preset loc 25 to show read or write, disc & mem addresses)*

		octal code	
a, #	WDi	64i00	Write data on MT unit i; i=Z(for 0), 1, 2, 3.
	WMi	64i20	Write marker from last 8 bits of (S) on MT unit i; i=Z(for 0), 1, 2, 3.
a, #	RDi	65i00	Read data from MT unit i; i=Z(for 0), 1, 2, 3.
	SMi	66i00*	Search for marker in last 8 bits of (S) on MT unit i; i=Z(for 0), 1, 2, 3.
	RWi	66i01	Rewind tape on MT unit i; i=Z(for 0), 1, 2, 3.
	BCK	60040	Backward. Perform operation in backward direction.
	NST	65004	No store. Do not store to memory. This is meaningful only for read MT orders.

\*Search is overlapped with computer operation, but next order to searching transport will hang until search is complete.

Oscilloscope and strip chart plot

octal code

PLT	67000	Plot on oscilloscope or strip chart.
ADV	67700	Advance movie film.

• Class 7, Analog Input, Shifts, Delays

Any Class 7 mnemonic may be followed by  $\rightarrow$  or  $+1$ , to cause storing of the final (U) at location M; by  $+2$ , storing (U) at location (B6); or by  $+3$ , storing (U) at  $M+(B6)$ . This class deals with various instructions used in conjunction with operation of the analog-to-digital converter.

The Class 7 mnemonics are as follows:

	octal code	
WAT	71100	Wait. Machine will <u>wait</u> until the next pulse from a crystal-controlled 1 kc. pulse generator before exiting Field 2.
LS1	72010	Special fast arithmetic shifts of double-length (U,R), left if S exponent positive, right if S exponent negative. Shifts are 8 bits at a time. LSi indicates i shifts of 8 bits. These shifts are principally used in unpacking converted data. The mnemonics may be combined to get different length shifts: LS4,LS1 would give 5 shifts of 8 bits (total: 40 bits). These shifts do not pass through the exponents of U or R nor through the sign of R, but do shift into the sign of U.
LS2	72020	
LS4	72040	
MCN	72110	Manual conversion. An A-to-D conversion of the channel specified by (S) will be performed.
ACN	72364	Automatic conversion. Six conversions from channels 1 through 6 will be performed.

Conversion results will be packed into U as follows: The 8 bits (sign plus 7 bits) resulting from each conversion will be packed into the mantissa with the bits resulting from the first conversion farthest to the left and the bits resulting from last conversion in the right-most 8 bits of U. The U exponent will be set to 77. The R mantissa is used.

There are sixteen channels into the converter. The channel to be converted is specified by the right-most 16 bits of S. Channel 1 corresponds to  $S_{m47}$ , Channel 2 to  $S_{m46}$ , etc.

In addition to the formal store options, operations may be performed with the 72xxx orders as follows:

72xxx + 400

(S) will be sent to U before performing any other operation.

72xxx + 200

(S) will be cleared and a 1 sent to  $S_{m47}$ .

72xxx + 4

(S) will be logically shifted 1 to the left each time (U,R) is shifted 8 to the left. Notice that this feature can be used to sample consecutively numbered channels automatically.

- Summary of Operation Codes

The accompanying chart summarizes the Field 2 mnemonics available in AP1 and AP2. If an operation code is followed by the symbol "@", the corresponding mnemonic causes IM bit 1 to be set to 1.

The symbol "→" following an operation mnemonic of class 1, 2, 5, 6, 7 causes a final store of U to M.

The symbol "-" preceding a class 5 operation mnemonic causes a final logical complement of U.

For more than one operation mnemonic in an instruction, the octal codes will be combined by a logical OR. In most cases, mnemonics are separated by commas. In class 0, the tests are separated by "+" for "ANY", by "x" for "ALL". The mnemonics "POS" and "NEG" are compound "ANY" tests and the mnemonics "PNZ" and "NNZ" are compound "ALL" tests.

## SUMMARY OF OPERATION CODES

## CLASS 0

HTR 00000@	IF(ANY)HTR 00000@	IF(ALL)HTR 04000@
TRA 01000@	IF(ANY)TRA 01000@	IF(ALL)TRA 05000@
SKP 02000	IF(ANY)SKP 02000	IF(ALL)SKP 06000
JMP 03000	IF(ANY)JMP 03000	IF(ALL)JMP 07000
PSN 00100	ZER 00010	TGi 0000i
MOV 00200	EVN 00020	NTG 00004
EOV 00300	SLN 00030@	NTi 00004+i
NSN 00500	NUL 00040	i=1,2,3
NMO 00600	NZE 00050	POS 00110
NEO 00700	ODD 00060	PNZ 00150
	SLF 00070@	NEG 00510
		NNZ 00550

## CLASS 1

ADD 10000	FAD 10400
SUB 10100	FSB 10500
MPY 10200	FMP 10600
DIV 10300	FDV 10700
BUS 14100	BSF 14500
IMP 10220	VDF 16700
IDV 13300	
VID 16300	
VDI 17300	

## CLASS 2

STO 20001@	RPL 20301
FST 20041	RPE 20701
BEU 21000	RPM 21001
BLU 21400	RPR 21401
BAU 20100	RPA 21601
BRU 20300	RPI 21501
BMU 20700	RWT 21641
BIU 20200	STi 200i0
CLA 21700	i=1,2,3
DBL 20004	WTG 20040
	NOP 30000

## CLASS 4

TSR 40000@	SLN 42000@	DMR 44000@
SBi 4000i@	ILN 42001@	DML 44010@
SPF 40007@	MLN 42002@	LUR 45010@
ACC 41000@	TLN 42003@	LUL 45020@
ABi 4100i@	SLF 42004@	LRR 45001@
APF 41007@	ILF 42005@	LRL 45002@
ERM 40020	MLF 42006@	LRS 45015@
i=1,...,6	TLF 42007@	LLS 45062@
BCT 46000@	STX 43005@	CRR 45055@
TFU 47000	STT 43006@	CRL 45066@
	SFT 43007@	

## CLASS 6

RTR 60000@	PRN 61110@	WDi 64i00
RHX 60100@	PRA 61210@	WMi 64i20
PHX 60400	PRO 61310@	RDi 65i00
PH7 60500	SPA 61010	NST 65004
PTR 60600	SPi 610i0	SMi 66i00
	i=2,...,6	RWi 66i01
TYP 60700	PAG 61070	BCK 60040
	PLT 67000	i=Z,1,2,3
	ADV 67700	

## CLASS 5

LDR 50400
LDU 50410
LTi 504i0
i=4,5,6,7
STF 50540
SUR 53000
XUR 54000
CPL 50100
ORU 50010
AND 50314
SYD 53220
SYS 53120
XTR 50020

## CLASS 7

WAT 71100
ACN 72364
MCN 72110
LSi 720i0
i=1,2,4

The tables on this page summarize the options available in SETU (Field 1), AUX (Field 3), and ADDR+MOD (Field 4). In the tables

A indicates the full length special registers Z, U, R, S, T4, T5, T6, T7 specified in the second triad by 0, 1, 2, 3, 4, 5, 6, 7.

B and Bi indicate the short index registers CC, B1, B2, B3, B4, B5, B6, PF specified in the second triad by 0, 1, 2, 3, 4, 5, 6, 7.

I and M indicate the number formed in the address field of the instruction. (M) indicates the contents of the memory location numbered M.

Exceptions are R→Z, 10 in field 3 and I or |Z|, 20 and -I or -|Z|, 30 in field 1. R→Z has the result that R is cleared to Z. I or |Z| has the result that an integer 1 goes to U. -I or -|Z| has the result that an integer -1 goes to U.

1st Triad		Field 1		1st Triad		Field 3	
(SETU)				(AUX)			
A	0	B	4	U→A	0	U→Bi	4
-A	1	-B	5	R→A	1	R→Bi	5
A	2	B	6	Bi+1	2	Bi-1	6
- A	3	- B	7	Bi+X	3	I→Bi	7

1st Triad		Field 4	
(ADDR+MOD)			
(M)	0	M	4
-(M)	1	-M	5
(M)	2	M	6
-  (M)	3	- M	7

Pseudo-orders govern the process of AP1 assembly and facilitate the handling of blocks of various types of data within AP1 programs. Pseudo-orders do not exist in AP2.

• ORG and END

All programs to be assembled by AP1 must be started by an ORG (origin) order and terminated by an END order.

The function of ORG is to initialize the assembly process, to identify the program which follows, and to determine whether it is to be assembled in relative or absolute final form. The ORG order is preceded by a "cr" and an "uc" or "lc" punch (upper or lower case).

A relativized program will run anywhere in memory. If an order in location P refers in Field 4 to location Q, it is through a Control Counter reference of the form  $CC+(Q-P)-1$ . A relativized program that will load under SPIREL control is generated if the LOCN field of the ORG pseudo-order is not blank; the ADDR field must be blank or zero in this case. To assemble a program to load with codeword at address N (octal) the ORG pseudo-order has the form

N		ORG
cr	1st tab	2nd tab

To assemble a program to load symbolically with name S (5 or fewer characters) the ORG pseudo-order has the form

S		ORG
cr	1st tab	2nd tab

To assemble a program to load as the Ath element of the Bth element ... of array K the ORG pseudo-order has the form

K,...,B,A		ORG
cr	1st tab	2nd tab

Here A,B,... are octal numbers; K is the codeword address or name



(as above) of the array to which the program belongs. As many as five levels may be specified. All control words are provided for the loading of the program as the designated array element.

A relativized program is also produced if the ORG pseudo-order has zero ADDR field and blank LOCN field. This form is only appropriate if the self-loading option is to be used during assembly. The self-loading tape produced will load with the LOAD switch beginning at the address in B6.

An absolute program will run only at the specified memory location. Field 4 reference to location Q is made directly. An absolute program is generated if the ADDR field is not blank or zero; the LOCN field must be blank or zero. To assemble a program to load at address M (octal) the ORG pseudo-order has the form

		ORG		M
cr	1st tab	2nd tab	3rd tab	

The program will load with the LOAD switch if the self-loading option is used during assembly; otherwise it will load under SPIREL control.

The END order has the form

		END		cr		cr
cr	1st tab	2nd tab				

where "END" must be immediately followed by two (or more) carriage returns.

Neither ORG nor END cause any words to be generated in a program.

- EQU

The EQU (equivalence) order gives a numeric equivalent for a symbol or equates one symbol to another. The order has the form

L		EQU		M
cr		1st tab		2nd tab   3rd tab

where L (in LOCN) is the symbol defined by the pseudo-order, SETU is blank, and M (in ADDR) is either absolute or a symbol whose value has previously been defined through its appearance in the LOCN field of another order. L is assigned the value M. If M is a 5-digit octal code, the symbol L may appear in the OPN field of any order following the EQU order; L will be treated as an operation code and will be replaced during assembly by the octal code for which it stands. No words are added to the program being assembled due to an EQU.

- BSS and BES

Either of these orders inserts a block of zero words into the body of the program. BSS (block started by symbol) and BES (block ended by symbol) have the form

L	XXX	M	
cr	1st tab	2nd tab	3rd tab

where L (in LOCN) is blank or symbolic, SETU is blank, and M (in ADDR) is absolute. M is the number of zero words to be inserted. If L is symbolic, it is assigned as if the LOCN field had been associated with the first (BSS) or last (BES) word in the block.

- BCD, FLX, REM

These orders deal with alphanumeric data and have the form

L		XXX		M
cr		1st tab		2nd tab   3rd tab

where SETU is always blank. The operation mnemonic must be followed by a "tab" character, and after that all characters (in the ADDR field M) are retained, 9 characters per word. Any occurrence of the "cr tab tab tab" sequence to continue the character string is replaced by a "space". For BCD (binary coded decimal), each character is converted to a corresponding printer hexad and the words are stored into the program being assembled; if L (in LOCN) is symbolic, it is assigned as if associated with the first word stored. For FLX (flexowriter), all codes (including case shifts, etc.) are preserved without conversion and the words are stored into the program being assembled; L (in LOCN) may be symbolic as for BCD. For REM (remarks), L (in LOCN) must be blank; this order is used only to obtain printed comments in the program listing, and no words are stored into the program being assembled.

- DEC, OCT, and HDC

The DEC (decimal), OCT (octal), and HDC (hexadecimal, i.e. base 16) orders are used for inserting numeric data into the body of the program. They have the form

L		XXX		M
cr	1st tab	2nd tab	3rd tab	

where L (in LOCN) is blank or symbolic, SETU is blank, and M (in ADDR) consists of a list of one or more octal or decimal numbers. If L is symbolic, it is assigned as if associated with the first number in the list. Each number must be separated from its successor by a comma, and each will be stored into a separate word in the program being assembled. Continuation lines should not be used; for long lists of numbers, several DEC or OCT pseudo-orders in succession may be used to produce a continuous block of data. An octal number consists of one to 18 octal digits. A decimal integer consists of one to 14 decimal digits; a floating point decimal number, of one to 14 significant figures and a decimal point. A hexadecimal number consists of one to 13 hexadecimal digits (0, 1, ..., 9, a, b, c, d, e, f). It may be 14 hexadecimal digits if its value is less than or equal to 3fffffffffffffff.

• REF

The REF (reference) order defines a single cross-reference word in the program being assembled. All REFs for a program must appear immediately after the ORG order, before any code for the program. The form of a REF order is

NAME		REF		CONTENT
cr	1st tab	2nd tab	3rd tab	

or

NAME		REF		*CONTENT
cr	1st tab	2nd tab	3rd tab	

Each REF must contain a location symbol, the name used to address it in the code for the program. The ADDR field of the REF specifies the content of the cross-reference word: a string of characters containing only upper case letters and numbers which will be converted to printer hexads, filled to 5 with '25' hexads or truncated to 5 as appropriate. If the cross-reference word is to contain an indirect addressing bit (for a vector, matrix or program), this is denoted by '\*' before the hexad string, with no intervening spaces or punches. If k REFs appear in a program, the first will be at location  $-(k-1)$  of the final program, ..., the  $k^{\text{th}}$  at location 77777 (-0). The punched output of the final program will be followed by a control word to set the initial index of the program to  $-(k-1)$ . When the program is loaded, execution of the control word to set initial index to  $-(k-1)$  will cause SPIREL to operate on each of the k cross-reference words as follows:

- 1) make an entry in the Symbol Table (ST) of the 5 hexads in the cross-reference word;
- 2) insert the corresponding Value Table (VT) address in the address field of the cross-reference word.

Indirect reference in the assembled program through the REF then causes addressing of the item with name in ST, the value in VT for a scalar or the codeword in VT for a vector, matrix, or program.

For a double operand, such as a complex scalar or non-scalar, two cross-references must be used and these must appear in the order of the parts of the operand. The name of the operand is associated with the first part, and the second part is named "ditto", which is printed '-----' but typed '#####'. If A is a complex scalar its cross-references might appear as

AREAL	REF	A
AIMAG	REF	-----
cr	1st tab	2nd tab   3rd tab

where '-----' is typed '#####'. It may be that one of the cross-references is never referred to in the code; this is the only case where an unlabelled REF may be used, but two REFs must be given.

• Application

Macro-orders are available in the APL assembly language. This facility allows the coder to define parameterized sequences of code and have these substituted in his program during assembly. Since a code pattern may thus be written only once for more than one occurrence in the program, a number of advantages are offered:

- Symbolic code for the program is shorter;
- code for the program is less prone to error because fewer instructions are prepared;
- the program is more easily changed because a single change in a macro definition will take effect in all occurrences at assembly;
- the program is more readable because single macro names appear in the code for operations which actually require sequences of machine instructions.

A macro-order is a general name which has been defined by the programmer to represent one or more valid APL instructions. Then, at each subsequent call of the macro-order, these instructions are inserted into the assembled program. Any order included in the macro-definition may contain a parameter in one or more fields; such a field may be changed each time a macro-order is called by specifying a different value for the parameter at each call.

Example. Suppose in an APL program there existed the following code:

CLA	ALPHA
FAD	B6+1,U→T4
STO	GAMMA
⋮	
CLA	B6
FAD	BETA,B6+1
STO	B6
⋮	
CLA	ALPHA
FAD	BETA,U→R
STO	GAMMA



The programmer could have saved himself the effort of writing the repetitious sequences of instructions by defining a macro-order called SUM with four parameters as follows:

```
SUM          MACRO      ADONE+ADTWO→TOTAL,AUX
              CLA       ADONE
              FAD       ADTWO,AUX
              STO       TOTAL
              MEND
```

Then, having defined the macro-order SUM, the programmer could call it in his APl code, using different parameter values at each call:

```
          SUM          ALPHA,B6+1, GAMMA,U→T4
          :
          :
          SUM          B6,BETA,B6,B6+1
          :
          :
          SUM          ALPHA,BETA,GAMMA,U→R
```

The instructions assembled would be identical with those originally written by the programmer, but the repetitious code would not appear in the program.

• Definition

A macro-definition specifies a set of instructions, gives the set a name, and determines which fields (if any) are to contain parameters. The macro-definition consists of three parts: (1) the MACRO pseudo-order, in which the LOCN field gives the name of the macro-order and the ADDR field gives the list of parameters; (2) the set of instructions to be represented by the macro-name; (3) the MEND pseudo-order, ending the macro-definition.

(1) The MACRO pseudo-order may or may not include a list of parameters and must be one of the following forms:

NAME		MACRO		PARA,PARB,...,PARZ
cr	1st tab	2nd tab	3rd tab	

NAME		MACRO	
cr	1st tab	2nd tab	

The name of the macro-order may be any valid API general name. This is its only appearance in the LOCN field; it is written in the OPN field at each call of the macro. If the macro-order has parameters, they are listed in the ADDR field of the MACRO pseudo-order. A parameter name is any valid API general name, and is separated from the next parameter name by one of the following special characters:

, = → + - × / ( )

The last parameter is followed by a carriage return; if more than one line is required, the 'cr tab tab tab' sequence follows (but does not replace) the separating character at the end of the first line. Note that if parentheses are used, they must be used in pairs. In this way meaningful notation may be employed in the list of parameter names; for example,

COMP		MACRO		RATE,TIME,DIST,TOTAL
------	--	-------	--	----------------------

could also be written

COMP	MACRO	RATE(TIME) → DIST, TOTAL
or		
COMP	MACRO	RATE × TIME = DIST - TOTAL

(2) Any reasonable number of instructions may be represented by the macro-name; generally, a lengthy set of instructions will best be coded in closed subroutine form rather than in the open form generated by a macro-order. Any valid APL instructions except pseudo-orders may be included. Symbols which have appeared in the ADDR field of the MACRO pseudo-order are parameters and are subject to the special rules described below; all other symbols are treated in accordance with the usual APL conventions. Orders within a macro-definition may conform to the rules for instruction content, or they may include parameter names which are then subject to the rules below.

LOCN: Symbolic LOCN fields which are not parameters may be used within a macro-definition, but such symbols are not meaningful outside the set of instructions comprising the macro-definition; they may be referenced only by other orders within the set. A symbolic LOCN field which is a parameter name must be given a different value at each call of the macro-order; these values may then be addressed by orders outside the macro-definition. Note, however, that orders within the macro-definition may reference LOCN symbols which appear elsewhere in the program, including those defined by pseudo-orders.

SETU: A single parameter name may appear in SETU, with or without the minus and absolute value signs normally permitted in this field. All values taken by this parameter at subsequent calls of the macro must then be valid SETU symbols or octal equivalents. Note that if a - or | | sign is included, it is effective regardless of whether another - or | | sign is used with a SETU

symbol as a parameter value at a subsequent call; such inflection signs are combined by a logical 'or'. If, at a given call, a SETU parameter value is omitted, it is replaced by the octal code '01' (do not change U).

OPN: Multiple parameter names are permitted in OPN to allow flexible coding of Class 0 tests, Class 2 tag orders, etc. These parameter names may be combined with the special symbols such as  $\rightarrow$ , +,  $\times$ , etc., normally permitted in this field. In the case of multiple parameters, values need not be specified for all parameters at every call if the resulting code is valid. Parameter values for OPN may include any valid OPN symbols or octal codes; the special symbols  $\rightarrow$ , +,  $\times$ , etc. may also be used as part of parameter values.

ADDR+MOD: This field may consist of a single parameter name, which is to assume a value equivalent to any valid ADDR+MOD form (e.g., \*ZETA, B1+B2+1, M+B6); or the field may include several parameters, provided the values they assume at any given call result in valid code (for example, SYMB+BREG+NUMB might become BETA+PF+3 or \*ALPHA+B2+1); or one or more parameter names may be combined with other symbols and/or numbers which are to remain the same at each call (such as NAME+B1+1, which might become ABC+B1+1 or XYZ+B1+1). A parameter value may be omitted entirely at a given call if such an omission does not destroy the validity of the remaining code. The special symbols such as \*, a, -, and | | may appear either with the parameter name or as part of the parameter value, and are combined by a logical 'or'.

AUX: This field may consist entirely of a single parameter name; if so, the value assumed by this parameter must be a valid AUX octal code or symbolic equivalent (e.g. U $\rightarrow$ T4, B1-1, etc.). Alternatively, either or both of the fast register symbols (and also I and X) may be represented by parameter names, provided that only valid combinations are used for parameter values (for example, B1-X and I $\rightarrow$ T4 are not permitted).

TAG: The customary TAG symbols (TG1, TG2, TG3) may appear within a macro-definition, or this field may contain a parameter name for which one of the above symbolic values will be substituted when the macro-order is called.

(3) The MEND pseudo-order which terminates the macro-definition is as follows:

MEND

| cr            | 1st tab    | 2nd tab

More than one macro-definition may appear within a given program, provided each is bracketed by its own MACRO and MEND pseudo-orders. The same parameter names may be used in separate macro-definitions without causing confusion, but they must not be used as symbols elsewhere in the program. A macro-definition may appear at any point in a program; it generates no code at this point, and transfers around the macro-definition are not needed. The only restriction is that a macro-order must be defined before it is called. One macro-definition may not appear within another, but a previously defined macro-order may be called within the definition of another macro-order.

• Call

After a macro-order has been defined, it may be called by writing the name of the macro-order in the OPN field of an instruction; if the macro-order uses parameters, their values for this particular call are listed in the ADDR field of the same instruction. Parameter values for a macro-order are listed in the same order as the list of parameter names in the MACRO pseudo-order of the corresponding macro-definition. Parameter values are separated by commas; the list is terminated by a cr, and the 'cr tab tab tab' sequence following a comma may be used to continue the list onto a second line. Certain parameters may be omitted at a given call; in this case, two adjacent commas (with or without spaces between them) or a comma followed by a cr indicate an omitted parameter. A macro-order will usually be called at several different points in a program. Any call may have a symbolic LOCN field, but no two calls may have the same symbolic LOCN field. The LOCN symbol is assigned to the first order of the set of instructions represented by the macro-order, unless the LOCN field of this order contains a parameter name for which a value is specified at the current call; in this case, the parameter value takes precedence. Note that several orders may replace a single macro-order; hence relative addressing around a call must be used with care.

At each call, the sequence of parameter values must correspond to the sequence of parameter names which appeared in the macro-definition, but the values assumed by the parameters will usually differ from one call to another. A parameter value may consist of any string of characters which, when substituted into the macro-definition at each occurrence of the corresponding parameter name, will produce valid API code for the field in which it occurs. If the call lies within another macro-definition, a parameter name from the outer macro-definition may be used as a parameter value for the inner macro-call.

• Examples

Suppose an APl program contains the following code:

B1	SB1	B2,U→B2
	LT4	*MATR1
B1	SB1	B2,U→B2
	⋮	
B3	SB3	B4,U→B4
	LT5	*MATR2
B3	SB3	B4,U→B4

This could be written by defining a macro-order such as

BREGS	MACRO	BA,BB,SBA,LTJ,MATRI
BA	SBA	BB,U→BB
	LTJ	*MATRI
BA	SBA	BB,U→BB
	MEND	

and calling it as follows:

BREGS	B1,B2,SB1,LT4,MATR1
⋮	
BREGS	B3,B4,SB3,LT5,MATR2

Another example of a macro-definition might be:

STORE	MACRO	TREG,OPN,TAG,SYMB,BMOD
TREG	OPN,TAG	SYMB+BMOD,I→BMOD
BMOD	RPA,WTG	SYMB-1
	MEND	

where the call

STORE	T4,STO,ST2,ALPHA,B3
-------	---------------------

would produce

T4	STO,ST2	ALPHA+B3,I→B3
B3	RPA,WTG	ALPHA-1

and the call

```

                                STORE      -T6,FST,B6,B1
would produce
    -|T6|      FST      B6+B1,I→B1
    B1         RPA,WTG    B6-1

```

All of the preceding examples are crowded with parameters in order to demonstrate the versatility and flexibility of macro-orders. In actual practice, many instances will be found where only one or two symbols vary at each repetition of otherwise identical blocks of code. Here the saving in programming time and in reducing the likelihood of introducing errors when copying lengthy sections of code will prove substantial. For example, the following block of code might occur repeatedly in a control program linking various subroutines:

```

LITES      MACRO      SUBR
            CLA        SL
            RWT        RESET
            SLF        77777
            TSR        *SUBR
            SLF        77777
RESET      SLN        (Z)
            MEND

```

Once defined, the macro-order "LITES" could be called at each point in the program where a transfer to a subroutine occurs. By specifying the particular subroutine as a parameter value of the macro-order, one order could be written in place of six each time.

A macro-order using no parameters at all would be useful, for example, in reversing the indexing of a matrix:



TRANS	MACRO	
B1	SB1	B2,U→B2
	LT4	*MATR
B1	SB1	B2,U→B2
	MEND	

At each call, the macro-order "TRANS" would cause T4 to be loaded with the desired element of the transposed matrix MATR.

As noted above, one previously defined macro-order may be called within the definition of another, producing a set of "nested" macro-orders. In the following example, such a set of nested macro-orders is used to multiply two matrices and store their product as a third matrix.

The outermost macro-order MULT has as parameters the codeword addresses and dimensions of the matrices involved; MATA has NROW rows and L columns, MATB has L rows and MCOL columns, and the product matrix MATC has NROW rows and MCOL columns. Within the initialization and storage operations performed by MULT, a second macro-order PROD is called; its definition uses two of the same parameters used by MULT and it performs the actual arithmetic and indexing operations required for the matrix multiplication. Both these macro-definitions are assumed to be embedded in a larger program in which numerous matrices of varying dimensions must be multiplied together.

PROD	
LOOP	B2
	B2
definition	B1
of	
inner macro	T4

```

graph TD
    MULT --> OUTER
    OUTER --> INNER
    INNER --> Z
    subgraph Definition
        direction TB
        D1[definition of outer macro]
        D2[call of inner macro]
    end
    subgraph Call
        direction TB
        C1[Z]
        C2[T5]
        C3[B2]
        C4[B1]
    end
    D2 --> C1
  
```

C

## ASSEMBLY PROCEDURE

An API program is assembled by exercising option #6 in the PLACER system.

Assembly output on the printer consists of error messages, program listing, and symbol table. These are discussed below. Assembly also provides a punched paper tape which contains the assembled program to be loaded under SPIREL control or with the LOAD switch. Assembly options are also discussed below.

Error indications. An API error indication is produced by apparent errors in syntax or sequencing. The type of error and its location are given by a message:

ERROR IN [F] AT CR NO [N]

where F is the name of the field in error

and N is the placer listing carriage return number of the line containing the error.

If a single instruction is continued onto more than one line, the carriage return number for the last line will pertain to the entire instruction.

Assembled program listing. Four columns are printed, giving:

- (a) The symbolic location (if any exists).
- (b) The location count, relative position of the word in the program, in octal.
- (c) The instruction in octal, broken into fields, with tag.
- (d) The symbolic address (if any exists).

Locations not assigned by the coder are assigned by the assembly program beyond the code for the program being assembled. These appear with their names below a row of asterisks in the program listing. A name may be one supplied by the coder, as 'A' in the case

STO      A

where 'A' never appears in a LOCN field. A name may also be one supplied by the assembly program for long octal or full length decimal numbers referenced in ADDR, as in the cases

```
                AND      77777 0000 7777 00000
or              CLA      d3.0
or              ADD      d412697
```

Specifically, the names assigned to numbers by the assembly program are '←0000A', '←0000B',... in order of occurrence in the program being assembled.

Symbol table. The table of symbols is printed out in seven columns giving information relevant to the symbols defined in the program:

- (a) The relative position in the table.
- (b) The symbol.
- (c) A number (usually 0) which determines the type of object for which the symbol stands.
- (d) The equivalent assigned to the symbol (5 octal digits), unless the symbol is a macro name or a macro parameter.
- (e) A number (usually 1) which indicates reference in the program to the symbol. A number 3 denotes a symbol which appears in a LOCN field but not in an ADDR field, so this may be an unnecessarily defined location in the program. A number 0 appears on macro names and macro parameters and on symbols given a numeric equivalent.
- (f) An 18 digit octal number. The first 5 digits indicate the line at which an equivalent was assigned.
- (g) A number which indicates how (if at all) the equivalent was assigned:
  - 0: by appearing in the LOCN field of an order.
  - 1: by appearing in the LOCN field of an EQU pseudo-order in which the address was symbolic (see section on pseudo-orders).
  - 2: by appearing in the LOCN field of an EQU pseudo-order in which the address was numeric (see section on pseudo-orders).

Assembly Options. If only option #6 of PLACER is requested, the stop

(I): 06 HTR CC

occurs. In addition to sense lights 14 and 15 which are turned on automatically, other sense lights may be turned on for special forms of output.

SL9 on: Print with double (instead of single) spacing.

SL11 on: Do not punch assembled program.

SL13 on: Punch self-loading tape. The tape produced will load by using the LOAD switch on the console. An absolute program will load to the origin specified. A relativized program will load to the setting of B6. These program forms are discussed under the ORG pseudo-order.

• Storage Exchange

This program STEX handles dynamic storage allocation in SPIREL. If B1 = codeword address of array and B2 = length of array upon execution of STEX, space is taken, and B1 = first word address of block upon exit. A more detailed explanation of the use of this program may be found in the SPIREL literature. The remarks in the program serve to explain the program's operation.

<u>Lines</u>	<u>Comments</u>
2	This program has codeword address 154.
6	+2, store to B6 option on class 5.
13	EQU'ed name in field 4; only the first 6-hexads of any name are retained.
25	Decimal integer constant in ADDR; 'a' bit is generated automatically due to shift order in OPN.
37	Simple store option '→' on class 1 arithmetic order; store is to fast register T6.
46	R is cleared to zero in AUX by R → Z, <u>not</u> Z → R.
60	Increment of CC in AUX causes a skip.
65	-I in field 1 sets U to the integer -1.
100	Only AUX is used here; no operation is performed in OPN.
101	I → B3 means final address to B3 in AUX.
110	More than two B-mods in field 4.
131	Store ATR to memory in OPN, compound mnemonic.
137	+3, store to B6 + M option in OPN.
155	Control counter is incremented by contents of X register in AUX, causing a jump.
174	Long octal constant is used in ADDR and is stored at bottom of program.

<u>Lines</u>	<u>Comments</u>
224	T7 is restored from value stored on the B6-list.
227-230	Labelled long octal constants out of code sequence. The first will be right-adjusted, filled with leading zeroes to 18 octal places.
231	Binary coded decimal psuedo-order generates two words of hexads here.
232-240	Equated symbolic names.

154		ORG		1
		REM	STEX FOR SPIRFL	2
				3
				4
				5
	T7	LT7+2	B1,B6+1	6
	-Z	TRA	a*SAVE,U+R	7
	7	BAU+2	X,B6+1	10
		LDR	STORAG	11
	7	LLS	d15,U+T6	12
	7	BAU	FIRSTEX,U+T5	13
	7	BAU	aB1,U+T4	14
	P1	IF(ZER)TRA	REORG,R+Z	15
	T7	IF(NUL)TRA	TAKE	16
				17
		REM	INACTIVATE SPACE ADDRESSED BY	31
GIVE1		CLA	B1,U+T7	21
		IF(NUL)TRA	GIVE5,U+B4	22
		CRL	d15,R+B3	23
	T7	LUR	d24,U+B5	24
	P5	LUR	3,U+B5	25
		IF(NUL)TRA	GIVE2	26
		LDR	MASK2	27
	7	IF(NUL)SKP	T7	30
		AB4	B5,CC+1	31
		AB4	B5-1	32
GIVE2	7	BAU	a33+1,U+R	33
	7	BAU	a34	34
		IF(NEG)SKP	T5	35
	F	ADD+	T5	36
	P4	RPA	B1	37
	T7	AND	MASK1	40
		IF(NUL)TRA	GIVE3	41
	P1	STO	B5,B6+1	42
	P4	ADD	a33-1,U+B1	43
	7	BAU+2	a33,B6+1	44
		TRA	GIVE1,R+Z	45
GIVE3	7	LDR+	B1,R+B4	46
	7	LLS	d15,U+PF	47
	7	BAU	a34-1,PF+1	50
		IF(POS)SKP	T5,R+7	51
		TRA	GIVE4	52
	FF	LRS	d15,B4-1	53
	P	BAU	*STORAG,I+B3	54
		STO	B4	55
	P3	IF(NUL)TRA	CC+1	56
	P4	RPA	B3,CC+1	57
	P4	RPA	STORAG	60
GIVE4	7	BAU	a31	61
		IF(NZF)SKP	T4	62
		TRA	TAKE	63
GIVE5	-I	ADD+	B5-1,P1-1	64
		IF(NZF)TRA	GIVE1,R+Z	65
		CLA	B5-2,I+B6	66
		TRA	GIVE3,U+B1	67
				70
				71
		REM	ACTIVATE BLOCK OF LENGTH B2+1	72
				73



4/11/64 11.32

PAGE 2

TAKF	P2	IF(ZEP)TRA	ATAKE,R+Z	74
	7	BAU	a32+1,U+T7	75
	T6	IF(POS)SKP	T7	76
	7	TRA	ATAKE,U+B1	77
		NOP	a7,U+T6	100
		LDR	*STORAG,I+B3	101
	7	LLS	a15	102
		IF(POS)SKP	T7,U+P5	103
		TRA	REJRG	104
TAKF1	T4	STJ	B3,B3+1	105
TAKF2	P5	LRS	a15,R+B4	106
	P5	IF(ZEP)TRA	TAKE3	107
	P	STJ	B3+B2,I+B4	110
TAKF3	P4	RPA	STJRG	111
	T4	IF(ZEP)TRA	ATAKE,R+Z	112
	P3	TRA	ATAKE,U+B1	113
				114
		RE1	WRITE ACTIVE BLOCKS TO LOW ADDRESSES	116
REORG	P6	MLN	04000,U+T7	117
		STX	Z,I+B6	120
		SB3	*FIRSTEX,I+B4	121
		TRA	REJRG7	122
REORG1		CLA	*B4,U+B5	123
		CRL	a15,R+B1	124
	P6	IF(NZF)TRA	REJRG2	125
		SB3	B3+B1+1,I+B4	126
		TRA	REJRG6	127
REORG2		CLA	a35+B4	130
		RPA,WTG	*B4	131
		AND	MASK1	132
		IF(NZF)TRA	REJRG2,B3+1	133
		AB3	B1,B4-1	134
	P4	RPA	CC+1,P1+1	135
	I	AB4,ERM	B1+1,U+B5	136
		CLA,WTG+3	B5,B1-1	137
		TRA	REJRG7	140
REORG3		CLA,WTG+3	B4,B4+1	141
REORG4		SUR,LDR+3	B4,B4+1	142
		IF(NUL)TRA	REJRG5,B1-1	143
		50114	MASK2,R+B5	144
		IF(NUL)TRA	CC+1	145
	P	LUL	a9,CC+1	146
	P	LUL	a9,B5-1	147
		UMR	a36	150
		IF(PSN*ZER)TRA	CC+1,B5+1	151
		ADD	a35-1,U+B5	152
	P3	RWT	B5-1	153
REORG5	P1	IF(NZF)TRA	REJRG4,B3+1	154
REORG6	7	BAU	a34,CC+X	155
		IF(NEC)SKP	T4	156
	P6	STX	Z,U+PF	157
REORG7	P4	IF(NZF)SKP	*LASTEX	160
		TRA	REJRG2	161
		LDR	B4	162
	7	LLS	a15,U+B1	163
		IF(NUL)TRA	REJRG1	164
	P3	SUB	a31+B4,I+B4	165
		TRA	REJRG7,U+B6	166

4/11/66 11.32

			PAGE	3
REORG8	T7	MLF	04000,U+B6	167
	T4	IF(NN7)SKP	T5	170
	PF	ADD+	T4	171
		IF(SLF)SKP	00002	172
		TRA	REORG8	173
		CLA	000024010000000000	174
		BAU	NOTE,U+T7	175
		SLN	00002	176
		TSR	*XCWD	177
		SLF	00002	200
REORG9		LDR	G,R+B4	201
		CLA	B4,U+P5	202
	Z	LDR	MASK2,U+PF	203
REORG10	P5	IF(NZF)SKP	B4+1,B4+1	204
	T6	TRA	REORG11,U+B5	205
		CLA	B4+1	206
		LUR	OP7,U+B1	207
		CLA	B1+PF,U+PF	210
		IF(NUL)SKP	Z,PF=1	211
		APF	1	212
	PF	RPA	B4+1	213
		TRA	REORG10	214
REORG11	T4	IF(NZF)TRA	TAKE1,R+Z	215
	Z	TRA	TAKE2,U+B2	216
				217
ATAKE	IT61	LRS	B15	220
	R	RPL	STORAG	221
	P1	STX	B*36-1,U+T7	222
	Z	IF(ZER,NTC)TRA	B*UNSAVE,B6-1	223
	T7	LT7	B6-1,U+B1	224
	PF	AB6	B77776,U+CC	225
				226
MASK1		UCT	400000000	227
MASK2		OCT	777777777740077777	230
NOT		BCD	REORGANIZATION	231
G		EQU	125	232
XCWD		EQU	126	233
SAVE		EQU	136	234
UNSAVE		EQU	137	235
STORAG		EQU	100	236
FIRSTEX		EQU	101	237
LASTEX		EQU	102	240
		END		241
				242
				243

## STEY FOR SPIREL

1	07	50472	26	0002	00000	
2	10	01000	02	4400	00136	SAVE
3	00	20100	26	0000	77775	
4	01	50400	00	0000	00100	STORAG
5	00	45062	06	4000	00017	
6	00	20100	05	0000	00101	FIRSTE
7	00	20100	04	4002	00000	
10	41	01010	10	4001	00070	REORG
11	07	01040	00	4001	00047	TAKE
INACTIVATE SPACE ADDRESSED BY B1						
GIVE1	12	01	21700	07	0002	00000
	13	01	01040	44	4001	00041
	14	01	45066	53	4000	00017
	15	07	45010	45	4000	00030
	16	45	45010	45	4000	00003
	17	01	01040	00	4001	00004
	20	01	50400	00	0001	00167
	21	00	02040	00	0000	00007
	22	01	41004	20	4040	00000
	23	01	41004	00	4040	77776
GIVE2	24	00	20100	02	4010	00001
	25	00	20100	00	4020	00000
	26	01	02510	00	0000	00005
	27	02	10001	00	0000	00006
	30	44	21601	00	0002	00000
	31	07	50314	00	0001	00155
	32	01	01040	00	4001	00004
	33	41	20001	26	4100	00000
	34	44	10000	41	4010	77776
	35	00	20102	26	4010	00000
	36	01	01000	10	4001	77752
GIVE3	37	00	50401	54	0002	00000
	40	00	45062	47	4000	00017
	41	00	20100	27	4020	77776
	42	01	02110	10	0000	00005
	43	01	01000	00	4001	00006
	44	47	45015	64	4000	00017
	45	02	20100	73	0400	00100
	46	01	20001	00	4020	00000
	47	43	01040	00	4001	00001
	50	44	21601	20	0010	00000
	51	44	21601	00	0000	00100
GIVE4	52	00	20100	00	4002	00000
	53	01	02050	00	0000	00004
	54	01	01000	00	4001	00004
GIVE5	55	30	10001	61	0100	77776
	56	01	01050	10	4001	77732
	57	01	21700	76	0100	77775
	60	01	01000	41	4001	77755
ACTIVATE BLOCK OF LENGTH B2+1						
TAKE	61	42	01010	10	4001	00117
	62	00	20100	07	4004	00001
	63	06	02110	00	0000	00007
	64	00	01000	41	4001	00114
	65	01	20000	06	4000	00000
	66	01	50400	73	0400	00100
	67	00	45062	00	4000	00017
	70	01	02110	45	0000	00007
	71	01	01000	00	4001	00007
TAKE1	72	04	20001	23	4010	00000
TAKE2	73	45	45015	54	4000	00017
	74	45	01010	00	4001	00001
	75	02	20001	74	4014	00000
						TAKE3

TAKER	76	44	21601	00	0000	00100	STORAG
	77	04	01010	10	4001	00101	ATAKE
	100	43	01000	41	4001	00100	ATAKE
WRITE ACTIVE BLOCKS TO LOW ADDRESSES							
REORG	101	46	42002	07	4000	04000	
	102	01	43005	76	4000	00000	
	103	01	40003	74	4400	00101	FIRSTE
	104	01	01000	00	4001	00035	REORG7
REORG1	105	01	21700	45	0420	00000	
	106	01	45066	51	4000	00017	
	107	46	01050	00	4001	00002	REORG2
	110	01	40003	74	4012	00001	
	111	01	01000	00	4001	00025	REORG6
REORG2	112	01	21700	00	4140	00000	
	113	01	21641	00	0420	00000	
	114	01	50214	00	0001	00072	MASK1
	115	01	01050	23	4001	00005	REORG3
	116	01	41003	64	4002	00000	
	117	44	21601	21	0001	00001	
	120	20	41024	45	4002	00001	
	121	01	21742	61	0040	00000	
	122	01	01000	00	4001	00017	REORG7
REORG3	123	01	21742	24	0020	00000	
REORG4	124	01	53403	24	0020	00000	
	125	01	01040	61	4001	00010	REORG5
	126	01	50114	55	0001	00061	MASK2
	127	01	01040	00	4001	00001	
	130	02	45020	20	4000	00011	
	131	02	45020	65	4000	00011	
	132	01	44000	00	4000	00044	
	133	01	05110	25	4001	00001	
	134	01	10000	45	4040	77776	
	135	43	21641	00	0040	77776	
REORG5	136	41	01050	23	4001	77764	REORG4
REORG6	137	00	20100	30	4020	00000	
	140	01	02510	00	0000	00004	
	141	46	43005	47	4000	00002	
REORG7	142	44	02050	00	4400	00102	LASTEX
	143	01	01000	00	4001	00005	REORG8
	144	01	50400	00	0020	00000	
	145	00	45062	41	4000	00017	
	146	01	01040	00	4001	77735	REORG1
	147	43	10100	74	4022	00000	
	150	01	01000	46	4001	77770	REORG7
REORG8	151	07	42006	46	4000	04000	
	152	04	06550	00	0000	00005	
	153	47	10001	00	0000	00004	
	154	01	02070	00	4000	00002	
	155	01	01000	00	4001	00005	REORG9
	156	01	21700	00	0001	00034	*0000A
	157	01	20100	07	4001	00031	NOTE
	160	01	42000	00	4000	00002	
	161	01	40000	00	4400	00126	XCWD
	162	01	42004	00	4000	00002	
REORG9	163	01	50400	54	0000	00125	G
	164	01	21700	45	0020	00000	
	165	00	50400	47	0001	00022	MASK2
REORG10	166	45	02050	24	4020	00001	
	167	06	01000	45	4001	00007	REORG11
	170	01	21700	00	0020	00001	
	171	01	45010	41	4000	00033	
	172	01	21700	47	0202	00000	
	173	01	02040	67	0000	00000	
	174	01	41007	00	4000	00001	
	175	47	21601	00	0020	00001	
	176	01	01000	00	4001	77766	REORG10

REOR11	177	04	01050	10	4001	77671	TAKE1
	200	00	01000	42	4001	77671	TAKE2
ATAKE	201	26	45015	00	4000	00017	
	202	02	20301	00	0000	00100	STORAG
	203	41	43005	07	4500	77776	
	204	00	01014	66	4400	00137	UNSAVE
	205	07	50470	41	0100	77776	
	206	47	41004	40	4000	77776	
MASK1	207	00	00000	00	4000	00000	
MASK2	210	77	77777	77	7400	77777	
NOTE	211	61	44564	14	6405	55071	
	212	40	43505	65	5252	52525	
*****							
*0000A	213	00	00240	10	0000	00000	

314	SAVE	0	134	0	2430000000000000	0
315	STORAG	0	100	0	2450000000000000	0
316	FIRSTE	0	101	0	2460000000000000	0
317	REORG	0	101	1	1240000000000000	0
320	TAKE	0	61	1	7700000000000000	0
321	GIVE1	0	12	1	2300000000000000	0
322	GIVE5	0	55	1	6600000000000000	0
323	GIVE2	0	24	1	3500000000000000	0
324	MASK2	0	210	1	2350000000000000	0
325	MASK1	0	207	1	2330000000000000	0
326	GIVE3	0	27	1	5000000000000000	0
327	GIVE4	0	52	1	6300000000000000	0
330	ATAKE	0	201	1	2240000000000000	0
331	TAKE1	0	72	1	1100000000000000	0
332	TAKE2	0	72	1	1110000000000000	0
333	TAKE3	0	76	1	1140000000000000	0
334	REORG7	0	142	1	1650000000000000	0
335	REORG1	0	105	1	1300000000000000	0
336	REORG2	0	112	1	1350000000000000	0
337	REORG6	0	137	1	1620000000000000	0
340	REORG3	0	123	1	1460000000000000	0
341	REORG4	0	124	1	1470000000000000	0
342	REORG5	0	136	1	1610000000000000	0
343	LASTEX	0	102	0	2470000000000000	0
344	REORG8	0	151	1	1740000000000000	0
345	REORG9	0	163	1	2060000000000000	0
346	*0000A	0	212	1	2510000000000000	0
347	NOTE	0	211	1	2370000000000000	0
350	XCWD	0	124	0	2420000000000000	0
351	G	0	125	0	2410000000000000	0
352	REOR10	0	166	1	2110000000000000	0
353	REOR11	0	177	1	2220000000000000	0
354	UNSAVE	0	137	0	2440000000000000	0

• Matrix Inverse

This program computes the inverse and determinant of a real matrix and prints an error message if the matrix is singular. The method used is essentially in-place Gaussian reduction as described in "An Introduction to Numerical Mathematics", Stiefel, E.L., 1963, page 3. Each successive pivot element is the largest in absolute value of all the remaining choices in a given column. The result is a compromise between speed and accuracy. An  $n \times n$  matrix is numerically singular if the ratio of any two pivot elements exceeds  $10^6/n$ . The codeword address of the matrix to be inverted is in T7 on entry, the inverse is stored as USTAR (codeword address 10), and the determinant is output in T7. If the matrix is singular, T7 = 0 on exit.

Lines 11 to 36:

The fast registers are saved, the input matrix is copied if necessary, internal constants are computed, the row codewords are labelled, and DET is initialized.

Lines 37 to 61:

The next column is scanned for the largest element, the largest and smallest pivot are stored and tested.

Lines 62 to 101:

The exchange algorithm is now applied to USTAR, the non-scalar accumulator in Genie and the pivot element is multiplied into DET.

Lines 102 to 113:

The two appropriate row codewords and their back references are exchanged if necessary.

Lines 114 to 151:

The columns of the final inverse matrix are now sorted as necessary due to non-diagonal pivoting.

Lines 152 to 157:

This section of code causes printing of an error message.

<u>Lines</u>	<u>Comments</u>
2	This is a symbolically named program, INV.
4-5	Cross-reference words for named items referred to by INV.
7	Extra carriage returns and a remark in the code sequence.
11	Use of +2 store option in operation field, store to B6.
12	Minus inflection in SETU, compound test in OPN, use of EQU'ed name in address field. The 'a' bit is not required since TRA gives this inflection automatically.
15-16	EQU'ed name in address field, and REF'ed name in address field.
35	Decimal constant in address field will be stored at the bottom of the program.
41	Absolute value inflections in SETU and ADDR, and indirect addressing specified by '*' in ADDR.
46	'→' codes as a store to M, here MAXP; '+1' in OPN is equivalent.
66	Enter repeat mode option on set or add to B-register orders.
106	Use of more than one B-modifier in field 4, B1 + PF + M (M = 0).
127	Reset X register from number originally stored on B6-list.
154	The address part of this instruction or M was replaced by the contents of PF at the instruction on line 13. Anything in ( ) is ignored in assembly.
160	A decimal constant is defined and is stored at EPSLN.
162-165	'Z' with OCT causes zero to be stored at these locations.

<u>Lines</u>	<u>Comments</u>
166-171	EQU psuedo-orders assign numeric values to names.
173-174	The END pseudo-order terminates the code but generates no instructions. It is followed by two carriage returns.



4/11/66 16.12

PAGE 1

INV	ORG		1
			2
MCOPY	REF	*MCOPY	3
ERRP	REF	*ERRP	4
			5
	REM	INV(T7) → USTAR	6
			7
	Z	BAU+2	10
	-Z	IF(ZER,EOV)TRA	11
	PF	RPA	12
	Z	BAU	13
		IF(ZER)SKP	14
	T7	TSR	15
	Z	STO	16
		STO	17
		CLA	20
		CRL	21
	-B4	CPL	22
		FMP	23
		VDF	24
		STO	25
ROWSTO		LDR	26
		LLS	27
	P3	LRS	30
	F	STO	31
	P3	IF(POS)SKP	32
		TRA	33
	P4	LDR	34
	F	STO	35
INVLP	Z	STX	36
		SB2	37
SCAN	IT61	IF(POS)SKP	40
	P1	LT6	41
	P3	AB1	42
	P1	IF(PN7)TRA	43
	IT61	IF(ZER,EOV)TRA	44
		LT5→	45
	T5	IF(ZER)TRA	46
	T4	IF(PN7)SKP	47
FIRST	T5	STO	50
	T4	LT6→	51
	T6	IF(ZER)TRA	52
	T4	IF(NN7)SKP	53
	T6	STO	54
STEST		CLA	55
		FDV	56
		IF(NEG)SKP	57
		TRA	60
	P3	LT4	61
	T4	LT5→	62
		FDV	63
	T5	FMP→	64
	P4	SB3,ERM	65
	T4	FMP→	66
LOOP1	P4	SB2	67
	E1	IF(ZER)JMP	70
	Z	LT5→	71
		SB2	72
			73

4/11/66 16.12

PAGE 2

LOOP2		SB1	a*T7	74
	T5	FMP	*IUSTAR	75
		SB1	aB4+1	76
		FAD+	*IUSTAR,B2-1	77
	P2	IF(PN7)TRA	aLOOP2	100
	P4	IF(PN7)TRA	aLOOP1	101
	T7	SB4	a*T6,U-B3	102
		CLA	USTAR,U-B1	103
	P3	IF(NZF)SKP	aPF	104
		TRA	aTEST,PF-1	105
		CLA	b1+PF,U-B2	106
		LDR+	b1+B3,I-B3	107
	P3	STO	aB2,R-B2	110
	R	STO	aB1+PF,I-B3	111
	P3	STO	aB2,PF-1	112
TEST	PF	IF(PN7)TRA	aINVL	113
		SB3	aB4,I-B2	114
HUNT		LDR	b1+B3	115
	Z	LLS	a+15	116
		IF(ZEP)SKP	aB2,B3-1	117
	P3	IF(PN7)TRA	aHUNT	120
	P2	IF(ZEP)SKP	aB3+1	121
		TRA	aSWAP,B3+1	122
	P1	TRA	aFIX,U+PF	123
LAST		SB2	aB2-1,I-B3	124
	P2	IF(PN7)TRA	aHUNT	125
OUT		TRA	a*JNSAYE	126
		STX	aB6-1,B6-1	127
	PF	LT7	DET,U+CC	130
SWAP	P1	SB1	aB4,U+PF	131
EXLOOP		LDR	*IUSTAR	132
	P2	SB2	aB3,U-B3	133
	R	LDR+	*IUSTAR	134
	P2	SB2	aB3,U-B3	135
	R	STO	*IUSTAR,B1-1	136
	P1	IF(PN7)TRA	aEXLOOP	137
		CLA	PF+B3	140
		LDR	PF+B2	141
		LLS	a+15,U-R	142
		CRR	a+15	143
		STO	aPF+B3	144
FIX		LDR	PF+B2	145
		LLS	a+15	146
	P5	LRS	a+15	147
	R	STO	aPF+B2	150
	PF	TRA	aLAST,U-B1	151
SINGLR	Z	SB1	aUSTAR,U-B2	152
		TSR	a*STEX	153
PFSAVE	I	SPF	a(Z),U-B1	154
		TRA	a*ERRP,B1+1	155
	Z	STO	aDET	156
		TRA	aOUT	157
EPSLN		DEC	1000000,0	160
TWO47		UCT	052000000000000000	161
ERROR		UCT	Z	162
MINP		UCT	Z	163
MAXP		UCT	Z	164
DET		UCT	Z	165
USTAR		EQU	10	166

4/11/66 16.12  
STEX EQU  
SAVE EQU  
UNSAVE EQU  
END

135  
136  
137

PAGE 3  
167  
170  
171  
172  
173  
174

PROGRAM INV

4/11/66 16.14

MCCPY	77776	54	4256F	77	0400	00000	
ERPR	77777	75	44A1A	15	7400	00000	
INV(T7) → USTAR							
	1	00	20102	26	0000	77775	
	2	10	01310	02	4400	00136	SAVE
	3	47	21401	10	0001	00140	FFSAVE
	4	00	20100	53	0000	00007	
	5	01	02010	71	4000	00010	USTAR
	6	07	40000	42	4401	77767	MCCPY
	7	00	20001	00	4001	00143	MINP
	10	01	20001	00	4001	00143	MAXP
	11	01	21700	41	0002	00000	
	12	01	4506A	54	4000	00017	
	13	54	50100	00	4000	00000	
	14	01	10600	00	0001	00134	TWO47
	15	01	16700	00	0001	00132	EPSLN
	16	01	20001	00	4001	00133	ERROR
ROWSTO	17	01	50400	23	0002	00001	
	20	01	45062	45	4000	00017	
	21	43	45015	00	4000	00017	
	22	02	20001	21	4002	00001	
	23	43	02110	00	4020	00000	
	24	01	01000	00	4001	77771	ROWSTO
	25	44	50400	47	0001	00130	+0000A
	26	02	20001	00	4001	00126	DET
INVLP	27	00	43005	06	4000	00007	
SCAN	30	01	40002	71	4200	00000	
	31	26	02110	00	2400	00010	USTAR
	32	41	50460	43	0400	00010	USTAR
	33	43	41001	07	4000	77776	
	34	41	05150	00	4001	77773	SCAN
	35	26	01310	00	4001	00104	SINGLR
	36	01	50451	04	0001	00115	MAXP
	37	05	01010	00	4001	00002	FIRST
	40	04	06150	00	0000	00005	
	41	05	20001	00	4001	00112	MAXP
FIRST	42	04	50461	00	0001	00110	MINP
	43	06	01010	00	4001	00002	STEST
	44	04	06F50	00	0000	00006	
	45	06	20001	00	4001	00105	MINP
STEST	46	01	21700	00	0001	00105	MAXP
	47	01	10700	00	0001	00103	MINP
	50	01	02510	10	0001	00101	ERROR
	51	01	01000	00	4001	00070	SINGLR
	52	43	50440	41	1001	00103	+0000A
	53	04	50451	52	0400	00010	USTAR
	54	01	10700	04	0000	00005	
	55	05	10601	22	0001	00077	DET
	56	44	40020	06	4040	00000	
	57	04	10601	63	0400	00010	USTAR
LOOP1	60	44	40002	41	4200	00000	
	61	41	03010	64	0000	00007	
	62	00	50451	00	0400	00010	USTAR
	63	01	40002	00	4040	00000	
LOOP2	64	01	40001	00	4400	00007	
	65	05	10600	00	0400	00010	USTAR
	66	01	40001	00	4020	00001	
	67	01	10401	62	0400	00010	USTAR
	70	42	05150	00	4001	77772	LOOP2
	71	44	05150	00	4001	77765	LOOP1
	72	07	40004	43	4400	00006	
	73	01	21700	41	0000	00010	USTAR
	74	43	02050	00	4200	00000	
	75	01	01000	67	4001	00005	TEST



337	•0000A	0	156	1	17700000000000000	0
330	DET	0	155	1	17100000000000000	0
331	INVLP	0	27	1	35000000000000000	0
332	SCAN	0	21	1	37000000000000000	0
333	SINGLR	0	142	1	15000000000000000	0
334	FIRST	0	42	1	50000000000000000	0
335	STEST	0	46	1	54000000000000000	0
336	LOOP1	0	60	1	66000000000000000	0
337	LOOP2	0	64	1	72000000000000000	0
340	TEST	0	103	1	11100000000000000	0
341	HUNT	0	105	1	11300000000000000	0
342	SWAP	0	121	1	12700000000000000	0
343	FIX	0	135	1	14300000000000000	0
344	LAST	0	114	1	12200000000000000	0
345	OUT	0	116	1	12400000000000000	0
346	UNSAVE	0	137	0	17500000000000000	0
347	EXLOOP	0	122	1	13000000000000000	0
350	STEX	0	125	0	17300000000000000	0

4/20/66 14.39

PAGE 1

INV		ORG		1
		REM	BACK-TRANSLATION	2
L77776		REF	*MCOPY	3
L77777		REF	*ERRP	4
L1	Z	BAU+2	77775, B6+1	5
	-Z	01310	a4136, U+R	6
	FF	RPA	L144, R+7	7
	Z	BAU	T7, R+13	10
		IF(ZER)SKP	a10, I+B1	11
	T7	TSR	*L77776, U+B2	12
	Z	STJ	L153	13
		STJ	L154	14
		CLA	B1, U+R1	15
		CRL	17, R+14	16
	-B4	CPL	aZ	17
		FMP	L151	20
		VDF	L150	21
		STJ	L152	22
L17		LDR	B1+1, R3+1	23
		LLS	17, U+15	24
	P3	LRS	17	25
	R	STJ	B1+1, R1+1	26
	P3	IF(POS)SKP	a34	27
		TRA	L17	30
	P4	LDR	L156, U+PF	31
	R	STJ	L155	32
L27	Z	STX	7, U+T4	33
		SB2	PF, I+R1	34
L31	IT61	IF(POS)SKP	I*101	35
	P1	LT6	*10, U+B3	36
	P3	AB1	77776, U+T7	37
	P1	IF(PN7)TRA	L31	40
	IT61	01310	aL142	41
		LT5+	L154, U+T4	42
	T5	IF(ZER)TRA	L42	43
	T4	IF(PN7)SKP	T5	44
	T5	STJ	L154	45
L42	T4	LT6+	L153	46
	T6	IF(ZER)TRA	L46	47
	T4	IF(NN7)SKP	T4	50
	T6	STJ	L153	51
L46		CLA	L154	52
		FDV	L153	53
		IF(NEG)SKP	L152, R+Z	54
		TRA	L142	55
	P3	LT4	-L156, U+B1	56
	T4	LT5+	*10, R+B2	57
		FDV	T5, U+T4	60
	T5	FMP+	L155, B2+1	61
	P4	40023	a35, U+T6	62
	T4	FMP+	*10, B3+1	63
L60	P4	SB2	PF, U+R1	64
	P1	IF(ZER)JMP	T7, B4-1	65
	Z	LT5+	*10	66
		SB2	B3	67
L64		SB1	*7	70
	T5	FMP	*10	71
		SB1	B4+1	72
				73

4/20/66 14.39

PAGE 2

		FAD+	*10,B2=1	74
	F2	IF(PN7)TRA	L64	75
	P4	IF(PN7)TRA	L60	76
	T7	SB4	*6,U+R3	77
		CLA	17,U+R1	100
	F3	IF(NZF)SKP	a0F	101
		TRA	L103,PF-1	102
		CLA	PF+B1,U+B2	103
		LDR+	B1+B3,I+B2	104
	P3	STO	B2,R+B2	105
	F	STO	PF+B1,I+B3	106
	F3	STO	B2,PF-1	107
L107	PF	IF(PN7)TRA	L27	110
		SB3	B4,I+B2	111
L108		LDR	B1+B3	112
	Z	LLS	17	113
		IF(ZER)SKP	a32,B2=1	114
	P3	IF(PN7)TRA	L105	115
	P2	IF(ZER)SKP	a33+1	116
		TRA	L121,B3+1	117
	P1	TRA	L135,U+PF	120
L114		SB2	B2-1,I+B3	121
	P2	IF(PN7)TRA	L105	122
L116		TRA	*137	123
L117		STX	*36-1,B6=1	124
	PF	LT7	L135,U+CC	125
L121	P1	SB1	B4,U+PF	126
L122		LDR	*10	127
	P2	SB2	B2,U+B3	130
	F	LDR+	*10	131
	P2	SB2	B2,U+B3	132
	F	STO	*10,B1=1	133
	P1	IF(PN7)TRA	L122	134
		CLA	PF+B3	135
		LDR	PF+B2	136
		LLS	17,U+R	137
		CRR	17	140
		STO	PF+B3	141
L135		LDR	PF+B2	142
		LLS	17	143
	P5	LRS	17	144
	F	STO	PF+B2	145
	PF	TRA	L114,U+B1	146
L140	Z	SB1	17,U+R2	147
		TSR	*135	150
L144	I	SPF	Z,J+B1	151
		TRA	*L77777,B1+1	152
L146	Z	STO	L135	153
		TRA	L116	154
L150		OCT	03017204400000000000	155
L151		OCT	06300000000000000000	156
L152		OCT	07000000000000000000	157
L153		OCT	07000000000000000000	160
L154		OCT	07000000000000000000	161
L155		OCT	07000000000000000000	162
L156		OCT	01001000000000000000	163
		END		164
				165
				166