

HP 3000 Computer Systems

**SYSTEM
PROGRAMMING
LANGUAGE**

Pocket Guide

HEWLETT  PACKARD

From	To					
	Long	Real	Double	Integer	Logical	Byte
Long	—	REAL				
Real	LONG	—	FIXR FIXT			
Double	LONG	REAL	—	INTEGER	LOGICAL	
Integer		REAL	DOUBLE	—	LOGICAL	BYTE
Logical			DOUBLE	INTEGER	—	BYTE
Byte			DOUBLE	INTEGER		—

NOTE: LOGICAL (Double) leaves the 32-bit value on the stack so that a 32-bit divide (/) can be performed. If only 16 bits of the double are desired, use LOGICAL (INTEGER (double)).

LOGICAL EXPRESSIONS: no type mixing. TRUE = -1(%17777), FALSE = 0. (Odd is true, even is false.)

Rank 1 (primary): bit operations, expr in parens, functions, assignment in parens, NOT expr, log. variables, constants, strings.

Rank 2: * (multiply), ** (two word result), / (divide), // (two word dividend), MOD (modulo), MODD (two word dividend).

Rank 3: + (addition), - (subtraction).

Rank 4: =, <, >, <=, >=, <>. Relops. Used for logical or algebraic comparisons, result is TRUE or FALSE.

Compare bytes:

$\left\{ \begin{array}{l} \text{Byte pointer (index)} \\ \text{Byte array (index)} \end{array} \right\} \text{ relop } \left\{ \begin{array}{l} \text{byte pointer (index), (count)} \\ \text{byte array (index), (count)} \\ \text{*PB, (count)} \\ \text{"string"} \\ \text{(constant list)} \end{array} \right\} [,sdec]$	
--	--

Stack:

S-2	2nd add.
S-1	1st addr.
S-0	Count

Test bytes:

byte variable	{ <> }	{ ALPHA	{ ALPHA = 101-132 ₈ and 141-172 ₈ .
	= }	NUMERIC	NUMERIC = 60-71 ₈ .
		SPECIAL	SPECIAL = all others.

* means stacked address.

{ } means optional.

{ } means pick one.

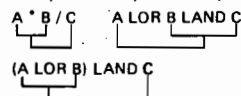
Sdec, see page 12.

Rank 5: LAND (conjunction, logical and, not the same as AND, see below.)

Rank 6: XOR (exclusive OR, true if either side is true, but not both.)

Rank 7: LOR (logical inclusive or, not the same as OR, see below.)

int expr <= int expr <= int expr (Compare range and branch; true if expr is within range.)



CONDITION CLAUSE: Used to make decisions in IF, WHILE, DO. Consists of logical expressions and/or branch words combined with the branch operators AND and OR (AND has higher precedence): CARRY, NOCARRY, OVERFLOW, NOOVERFLOW, <, =, >, <=, >=, <>, IABZ, DABZ (Incr/decr TOS, true if zero), IXBZ, DXBZ (same for X).

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

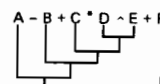
ARITHMETIC EXPRESSIONS: no type mixing across operators; use type transfer functions (page 2).

Rank 1 (primary): Bit operations, expr in parens, absolute value ($\backslash$ expr $\backslash$), functions, assignment in parens, constants, and variables.

Rank 2: $\wedge$ (exponentiate) (int to int, real to int, real to real, long to long, long to int).

Rank 3: $*$ (multiply), $/$ (divide), and MOD (modulo).

Rank 4: $+$ (add), $-$ (subtract).



CONSTANTS: "string of characters" 1243 (decimal),
%177 (octal) %(2)100101 (binary), %(16) AE (hex)
[number bits/constant ...]
Composite integer constant composed of bit fields.
integer constant D (DOUBLE)
fraction E $\pm$ power (real)
fraction L $\pm$ power (long)
composite E or L (not floated)

VARIABLES: All data types of simple variables and indexed pointers and arrays,
TOS (top of stack).

Integer only: @ simple variable, @ array (index); DB relative address.

@ pointer (index); DB relative address of object of pointer.

@ label, @ procedure, @ entry; PB relative address.

ABSOLUTE (address). Privileged access to an absolute location.

Notes: @ means "address of" for simple variable, array.

@ means "contents of" for pointer.

BIT FUNCTIONS: legal in ARITHMETIC and LOGICAL EXPRESSIONS.

One-word Extract: primary. (left extract bit : bit field length) Result right-justified. CAT extracts from second and deposits in first primary.

One-word Concatenate: primary CAT primary (left deposit bit : left extract bit : bit field length).

Shifts: primary & shiftop (count) (shiftops = ASL, ASR, LSL, LSR, CSL, CSR, DASL, DASR, DLSL, DLSR, DCSL, DCSR, TASL, TASR, TNSL.) No type checking by compiler.

IF EXPRESSIONS (logical or arithmetic): IF condition clause THEN expr ELSE expr.

COMMENT any characters ; <<comment text>>

An identifier consists of up to 15 alphanumeric characters, beginning with an alphabetic and containing optional apostrophes ('): AB, A23, AB'CES, A12345678901234.

All identifiers must be declared except labels. Declarations must occur before use and at the beginning:

program	{	BEGIN	data declarations {define, equate, simple variables, array, pointer, label, switch, entry.
			procedure and intrinsic declarations.
			subroutine declarations.
		END.	statements

DEFINE identifier = text #, ... ; Assigns text to identifier. Inserts text wherever identifier appears, at all points in the program.

DEFINE I = ARRAY B(0:1) #; INTEGER I; <<equals INTEGER ARRAY B (0:1);>>

EQUATE identifier = constant expression, ... ; Assigns constant to identifier. Replaces identifier with constant wherever the identifier appears.

EQUATE M = 1, N = M + 2, P = N * M; <<M = 1, N = 3, P = 3>>

Simple Variable:

$\left\{ \begin{array}{l} \text{INTEGER} \\ \text{BYTE} \\ \text{LOGICAL} \\ \text{REAL} \\ \text{LONG} \\ \text{DOUBLE} \end{array} \right\}$	$\left\{ \begin{array}{l} \text{identifier} [= \text{constant}] \\ \text{identifier} [= \text{identifier} [\pm \text{int}] \\ \text{register} \pm \text{int}] \end{array} \right\} \dots$
--	---

If equals (=) follows the identifier, variable reuses space of other variable.

If colon-equals (:=) follows identifier, next DB or Q cell is allocated and initialized.

If nothing follows, next cell is allocated, uninitialized.

Global: GLOBAL variable dec ; <<external proc. link>>

Local: OWN var. dec ; <<no references>>

EXTERNAL var. dec ; <<link for ext proc>>

Register = DB, Q, or S.

The general form of declaration is:

INTEGER
REAL
LONG
DOUBLE
BYTE
LOGICAL

ARRAY identifier (lower bound: upper bound) := [constant,] [repeat factor (constant, ...)] ... ;

(Logical is the default type for arrays. Global arrays are allocated space DB+ and local arrays Q+. Therefore, local arrays disappear when procedures exit. The bounds are integer constants specifying the upper and lower element numbers. Array elements are accessed by specifying an element index (expression or assignment) in parens after the array identifier. The identifier by itself specifies the 0 element.)

Options: Direct access instead of indirect through a pointer: identifier (bounds) = DB, or id (bounds) = Q.

OWN: local array allocated in DB area with Q+ pointer. Storage remains upon exit.

GLOBAL-EXTERNAL: links procedure compiled separately with data in outer block.

Unbounded arrays: * instead of bounds means a direct array or equivalenced.

@ instead of bounds means an indirect array. These arrays always reuse space allocated to other variables. Space is assigned by reference.

The table below shows most of the array types available in SPL/3000. Each type can be both global and local, unless specified otherwise.

WORD ADDRESS		BYTE ADDRESS	
DB relative address		DB relative address	B
0	15	0	14 15

B = 0 (left byte), 1 (right byte)

Convert byte address to word with ASR(1); word to byte with ASL(1).

Bounded arrays	Allocate array space?	Allocate a pointer cell?	Specify address of pointer explicitly?	Specify address of 0 element explicitly?	Can be initialized?
Indirect ARRAY BOX(0:25);	YES, new space	YES, in DB+ or Q+ Cell	NO, next available cell	NO, allocated in next available space	YES, if global
Direct ARRAY BOX(0:25)=DB; ARRAY BOX(0:25)=Q;	YES, new space	NO, 0 element is directly accessed	NO, no pointer exists.	NO, allocated in next available space	YES, if global
P Relative ARRAY BOX(0:25)=PB:=26(0);	YES, new space in code	If array is referenced out of P±255 range	NO, set in code if needed	NO, allocated in available code space	MUST BE completely initialized or size is reduced.
Local OWN OWN ARRAY BOX(0:25);	YES, new space	YES, Q+ cell	NO, next Q+ cell	NO, allocated in secondary DB	YES
Dynamic Local ARRAY BOX(M,N);	YES, allocated dynamically in procedure stack	YES, upon procedure entry	NO, allocated in Q	NO, array is allocated in available Q+	NO, space not allocated until proc. entry
Equivalenced or Unbounded Arrays					
Equivalenced ARRAY BOX(6)=RAR(7);	NO, re-uses space	DIRECT--NO ----- INDIRECT YES if needed	NO, if needed, pointer is allocated in DB+ or Q+	YES, by reference	NO, no new space

ASSIGNMENT STATEMENT

$\left\{ \begin{array}{l} \text{variable} \\ \text{variable. (left bit: length)} \end{array} \right\} := \left[\text{variable} := \dots \right] \text{expression};$

Deposits for single word variables only.

A := (B + C) / D;
A := IF N = 5 THEN C ELSE D;
A := B := C := 0;
A. (5 : 6) := %4;

CASE [*] expression OF compound statement;

First statement has index 0.

* means no bounds checking.

Note: Destroys Index Register.

CASE N OF

BEGIN A := 0;
B := A + 2;
C := 22 * B;
N := 12;

END;

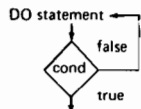
If the index expression is not within bounds, the next statement after the CASE is executed.

COMPOUND STATEMENT

BEGIN [statement ; ... [statement]] END;

DEL ; DEL deletes S-0.
DELB ; DELB deletes S-1.
DDEL ; DDEL deletes S-0 and S-1.

DO statement UNTIL condition clause;



DO A(I := I + 1) := I * 2 UNTIL I > 23;
DO BEGIN I := I + 1;
IVAL(I) := I / (X * Y + 3);
BCAL(I) := (X * Y + 3) / I;
END <<NO SEMICOLON>>
UNTIL I > 20;

Note: See CONDITION CLAUSE, page 3.

FOR [*] integer variable := expr [STEP expr] UNTIL expr DO statement ; <<expr must be single word>>

* means to execute the loop statement once before testing the loop variable against the limit.

STEP, if missing, is 1. Do not upset the stack within a FOR. Enter FOR only at top, leave only at end.

FOR I = MAX STEP (-RANGE/4) UNTIL MAX-RANGE DO

BEGIN FOFI := A * I ^ 2 + B * I + C;
SUM := SUM + FOFI;

END;

Stacked S-2
control S-1
values S-0

address
step
limit

Missing if loop = X.

$\left\{ \begin{array}{l} \text{GO} \\ \text{GOTO} \\ \text{GO TO} \end{array} \right\} \left\{ \begin{array}{l} \text{label} \\ \text{[*] switch (index)} \end{array} \right\}$

The index can be any one-word expression or assignment.

* means no bounds checking.

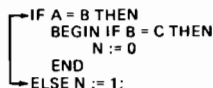
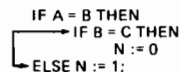
GO TO L1; GO SW1(N); GOTO * SW1(N + 3);

With bounds checking, if the index is not in bounds, the next statement is executed.

IF condition clause THEN statement [ELSE statement] ;

Notes: 1. No semicolon after THEN part and before ELSE part.

2. When nested, innermost ELSE is matched with the closest preceding THEN. Use BEGIN-END to override.



IF A = B OR CARRY THEN NX := A + B;

IF A <> B AND = THEN XA := XA + A ELSE XA := XA + B;

See CONDITION CLAUSE, page 3.



Move Words:

$$\text{MOVE } \left\{ \begin{array}{l} \text{non-byte pointer (index)} \\ \text{non-byte array (index)} \end{array} \right\} := \left\{ \begin{array}{l} \text{non-byte pointer (index), (word count)} \\ \text{non-byte array (index), (word count)} \\ * \text{ (word count)} \\ * \text{PB, (word count)} \\ \text{string} \\ \text{(constant list)} \end{array} \right\} [, \text{sdec}] ;$$

Destination → Source →

$$\text{MOVE } * := \left\{ \begin{array}{l} \text{non-byte pointer (index), (word count)} \\ \text{non-byte array (index), (word count)} \end{array} \right\} [, \text{sdec}] ;$$

Sdec = stack decrement
 0 leaves all values,
 1 leaves dest. source,
 2 leaves dest.
 Blank deletes all.
 * means DB address on the stack.
 (index) is optional.

Move Bytes:

$$\text{MOVE } \left\{ \begin{array}{l} \text{byte pointer (index)} \\ \text{byte array (index)} \end{array} \right\} := \left\{ \begin{array}{l} \text{byte pointer (index), (byte count)} \\ \text{byte array (index), (byte count)} \\ * \text{PB} \\ \text{string} \\ \text{(constant list)} \end{array} \right\} [, \text{sdec}] ;$$

Stacked values: S-2 Destination
S-1 Source
S-0 Count

Move Bytes While:

A means alphabetic,
 N means numeric,
 S means upshift.

$$\text{MOVE } \left\{ \begin{array}{l} \text{byte pointer (index)} \\ \text{byte array (index)} \end{array} \right\} := \left\{ \begin{array}{l} \text{byte pointer (index)} \\ \text{byte array (index)} \end{array} \right\} \text{ WHILE } \left\{ \begin{array}{l} \text{A} \\ \text{N} \\ \text{AS} \\ \text{AN} \\ \text{ANS} \end{array} \right\} [, \text{sdec}] ;$$

Sets CCB on last byte.

Stacked values: S-1 Destination
S-0 Source

$$\text{SCAN } \left\{ \begin{array}{l} \text{byte pointer (index)} \\ \text{byte array (index)} \end{array} \right\} \left\{ \begin{array}{l} \text{WHILE} \\ \text{UNTIL} \end{array} \right\} \text{testword } [, \text{sdec}] ;$$

Stack: S-1 Byte address
S-0 Testword

Testword =

0	7 B	15
terminal	test	

 CARRY set if terminal found, NOCARRY for test.

$$\left\{ \begin{array}{l} \text{PUSH} \\ \text{SET} \end{array} \right\} (\text{register list}) ; \text{Registers are S, Q, X, STATUS, Z, DL, DB.}$$

The order these registers are stacked before PUSH or after SET:

S	TOS-n
Q	.
X	.
STATUS	.
Z	.
DL	.
DB (absolute)	TOS

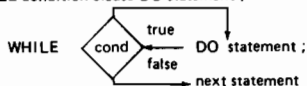
All address register values are relative to DB, except DB, which is an absolute address.

RETURN [count] ;
 <<Returns from proc or sub, deletes count parm words.
 If count omitted, deletes all parameters.>>

SUBROUTINE CALL:
 subroutine identifier [([* , ...] param, ... param)] ;

See PROCEDURE CALL below for param meaning.

WHILE condition clause DO statement ;



Notes: See CONDITION CLAUSE, page 3
 Reverse sense of these: IABZ, DABZ, IXBZ, DXBZ. While non-zero, do statement.

WHILE DXBZ DO A(IND) := 0;
 WHILE I < 21 DO A(I := I + 1) := 2 * I;

PROCEDURE CALL: procedure identifier [([* , ...] param, ... param)] ; * means param is stacked.

The params can be values (expr, assignments) or reference (simple variable, array, pointer, procedure, entry, label). Legal combinations of actual (calling) params and formal (declared) params are shown in the table on the next page. Empty entries in table are illegal.

PROC2 (*, *, RA) ; PROC2 (R9, A6, A := A + B) ;	Q-3 Q-2 Q-1 Q-0	Index Register Relative P STATUS Reg. Delta Q	} Stack Marker	Q-n-3 . Z Q-4	First param . . Last param	} Parameter Passing
--	--------------------------	--	----------------	------------------------	-------------------------------------	---------------------

Actual Parameters Passed to Formal Parameters

Actual Parameter	Formal Parameter						
	Simple Variables By Reference	Simple Variables By Value	Arrays	Pointer By Reference	Pointer By Value	Procedures	Labels
Constant	Warning (uses value of constant address)	Must be same word size.	Warning (uses value of constant address)	Warning (uses value of constant address)	Warning (uses value of constant address)		
Expression		Must be same word size.					
Simple Variable Identifier	OK	Must be same word size.	OK, loads address of simple variable		OK, load address of simple variable		

Array Reference	OK	Must be same word size.	OK		OK		
Pointer Reference	OK	Must be same word size.	OK	OK	OK		
Procedure Identifier						OK	
Label Identifier							OK
* (stacked)	OK	OK	OK	OK	OK	OK	

FOR 029 KEYPUNCH: ASCII [= 029 ϵ ASCII ^ = 029 $\neg$
 ASCII \ = 029 0-8-2 ASCII ! = 029 |
 ASCII] = 029 !

ASSEMBLE ([*label* :] *instruction* [; ... ; [*label* :] *instruction*])

label

identifies the instruction. The label is an SPL identifier.

instruction

indicates a machine instruction to be executed or a pseudo-op to generate a constant. The instruction conforms to one of the ten formats shown below.

EXAMPLES:

```
ASSEMBLE (LOAD A;
          L1: DUP,ZERO;
          STOR C;
          STOR D);
ASSEMBLE (LOAD P+ 0;ZERO;STD A);
```

NOTES:

I = Indirection

X = Index Register or Indexing

label id = A statement or instruction label within addressing range.

variable id = A data item identifier within addressing range.

usi = An unsigned integer less than or equal to the integer specified. For example *usi*255 means an unsigned integer between 0 and 255 inclusive.

Format 1

1a $\left\{ \begin{array}{l} \text{LOAD} \\ \text{LDX} \\ \text{LRA} \\ \text{CMPM} \\ \text{ADDM} \\ \text{SUBM} \\ \text{MPYM} \end{array} \right\} \left\{ \begin{array}{l} \text{label id} \\ \text{variable id} \\ \text{DB} + \text{usi255} \\ \text{P} + \text{usi255} \\ \text{P} - \text{usi255} \\ \text{Q} + \text{usi127} \\ \text{Q} - \text{usi63} \\ \text{S} - \text{usi63} \end{array} \right\} \quad [,\text{I}] \quad [,\text{X}]$

1b $\left\{ \begin{array}{l} \text{LDB} \\ \text{LDD} \\ \text{STOR} \\ \text{STB} \\ \text{STD} \\ \text{INCM} \\ \text{DECM} \end{array} \right\} \left\{ \begin{array}{l} \text{variable id} \\ \text{DB} + \text{usi255} \\ \text{Q} + \text{usi127} \\ \text{Q} - \text{usi63} \\ \text{S} - \text{usi63} \end{array} \right\} \quad [,\text{I}] \quad [,\text{X}]$

1c $\text{BR} \quad \left\{ \begin{array}{l} \text{label id} \\ \text{P} + \text{usi255} \\ \text{P} - \text{usi255} \end{array} \right\} \quad [,\text{I}] \quad [,\text{X}]$

$\text{BR} \quad \left\{ \begin{array}{l} \text{DB} + \text{usi255} \\ \text{Q} + \text{usi127} \\ \text{Q} - \text{usi63} \\ \text{S} - \text{usi63} \end{array} \right\} \quad \text{I} \quad [,\text{X}]$

1d $\left\{ \begin{array}{l} \text{BL} \\ \text{BE} \\ \text{BLE} \\ \text{BG} \\ \text{BNE} \\ \text{BGE} \end{array} \right\} \left\{ \begin{array}{l} \text{label id} \\ \text{P} + \text{usi31} \\ \text{P} - \text{usi31} \end{array} \right\} \quad [,\text{I}]$

1e $\left\{ \begin{array}{l} \text{TBA} \\ \text{MTBA} \\ \text{TBX} \\ \text{MTBX} \end{array} \right\} \left\{ \begin{array}{l} \text{label id} \\ \text{P} + \text{usi255} \\ \text{P} - \text{usi255} \end{array} \right\}$

where

variable id is a simple variable, pointer, or array identifier, (indirection is not supplied automatically).

usi is an unsigned integer less than or equal to the number following.

label id is a label which is used to label a statement within the range of the instruction.

For example,

ASSEMBLE(STB S - 1, I, X; DECM VAR);

Format 2

stackop
or
stack op, stack op

In the first case the compiler fills in the second half of the instruction word with a NOP.

The legal *stackops* are as follows:

NOP	DNeg	XCH	FLT	NOT
DELB	DXCH	INCA	FCMP	OR
DDEL	CMP	DECA	FADD	XOR
XROX	ADD	XAX	FSUB	AND
INCX	SUB	ADAX	FMPY	FIXR
DECX	MPY	ADXA	FDIV	FIXT
ZERO	DIV	DEL	FNEG	INCB
DZRO	NEG	ZROB	CAB	DECB
DCMP	TEST	LDXB	LCMP	XBX
DADD	STBX	STAX	LADD	ADBX
DSUB	DTST	LDXA	LSUB	ADXB
MPYL	DFTL	DUP	LMPY	
DIVL	BTST	DDUP	LDIV	

For example,

ASSEMBLE(DDUP, DELB; STAX);

Format 3

3a $\left\{ \begin{array}{l} \text{IABZ} \\ \text{IXBZ} \\ \text{DXBZ} \\ \text{BCY} \\ \text{BNCY} \\ \text{CPRB} \\ \text{DABZ} \\ \text{BOV} \\ \text{BNOV} \\ \text{BRO} \\ \text{BRE} \end{array} \right\} \left\{ \begin{array}{l} \text{label} \\ P \pm \text{usi}31 \\ * \pm \text{usi}31 \end{array} \right\} \quad [.J]$

3b

$\left\{ \begin{array}{l} \text{ASL} \\ \text{ASR} \\ \text{LSL} \\ \text{LSR} \\ \text{CSL} \\ \text{CSR} \\ \text{SCAN} \\ \text{TASL} \\ \text{TASR} \\ \text{TNSL} \\ \text{DASL} \\ \text{DASR} \\ \text{DLSL} \\ \text{DLSR} \\ \text{DCSL} \\ \text{DCSR} \\ \text{TBC} \\ \text{TRBC} \\ \text{TSBC} \\ \text{TCBC} \\ \text{QASL} \\ \text{QASR} \end{array} \right\}$

In these branch instructions, the address can be specified as a *label* or a P relative address ($P \pm *$ are the same thing). If the label location is not within 31 locations of P ($P \pm 31$), the compiler tags this as an error; indirection is *not* supplied automatically within an ASSEMBLE statement.

usi63 [.X]

usi63 is a shift count or number of bits less than or equal to 63. For example,

ASSEMBLE(LSL 1; BRE QUIT);

Format 4

4a $\left\{ \begin{array}{l} \text{LDI} \\ \text{LDXI} \\ \text{CMPI} \\ \text{ADDI} \\ \text{SUBI} \\ \text{MPYI} \\ \text{DIVI} \\ \text{PSHR}^\dagger \\ \text{LDNI} \\ \text{LDXN} \\ \text{CMPN} \\ \text{SETR}^\dagger \end{array} \right\}$

† = a privileged instruction for some registers

usi255

4b $\left\{ \begin{array}{l} \text{EXF} \\ \text{DPF} \end{array} \right\}$

usi15 : usi15

For example,

ASSEMBLE (LDI 255; ADDI 5; EXF 7:9);

Format 5

<pre> { RSW LLSH† PLDA† PSTA† LSEA† SSEA† LDEA† SDEA† IXIT† LOCK† PCN† UNLK† } </pre>	† = a privileged instruction
---	------------------------------

For example,

ASSEMBLE (RSW; PLDA; . . . LLSH; . . . PSTA);

Format 6

<pre> { PAUS SED XCHD SMSK RMSK XEQ SIO RIO WIO TIO CIO CMD SIN HALT LST PSDB DISP PSEB SCLK RCLK SST } </pre>	usi15
--	-------

For example,

ASSEMBLE (XEQ 4);

All of these instructions except XEQ and RMSK are privileged.

Format 7

<pre> { PCAL SCAL EXIT SXIT ADXI SBXI LLBL LDPP LDPN ADDS SUBS ORI XORI ANDI } </pre>	usi255
---	--------

PCAL *procedure identifier*

SCAL (user must load label onto stack)

LLBL *procedure identifier*

For example,

ASSEMBLE (PCAL READ; . . . SCAL LOOPER; . . . ORI %377);

Format 8

8a $\left\{ \begin{array}{l} \text{MOVE} \\ \text{MVB} \\ \text{CMPB} \end{array} \right\} [\text{PB}] \left[\begin{array}{l} .0 \\ .1 \\ .2 \\ .3 \end{array} \right]$

If item two is empty, a DB relative move is assumed.
If item three is empty, the stack decrement is 3.

8b $\text{MVBW} \left\{ \begin{array}{l} \text{A} \\ \text{N} \\ \text{AN} \\ \text{AS} \\ \text{ANS} \end{array} \right\} \left[\begin{array}{l} .0 \\ .1 \\ .2 \end{array} \right]$

If item three is empty, the stack decrement is 2.

8c $\left\{ \begin{array}{l} \text{MVBL}† \\ \text{MVLBT} \\ \text{SCW} \\ \text{SCU} \end{array} \right\} \left[\begin{array}{l} .0 \\ .1 \\ .2 \end{array} \right]$

†Privileged instruction.

If item two is missing, the stack decrement is 2.

For example,

```
ASSEMBLE (SCW, 1);
ASSEMBLE (MVBW AN, 0);
ASSEMBLE (CMPB PB, 1);
```

$$8d \left[\begin{array}{l} \text{MABS}\dagger \\ \text{MTDS}\dagger \\ \text{MDS}\dagger \\ \text{MFDS}\dagger \end{array} \right] \left[\begin{array}{l} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \text{ for MABS and} \\ \text{MDS} \end{array} \right]$$

Format 9

CON *constant list*

This format is actually a pseudo-mnemonic for constant generation; it is not a hardware instruction.

CON stores a series of constants in the code starting at the current location. In addition to all numerical and string constants, P relative address constants can be created by listing label identifiers (this is used to create address for indirect references). The CON instruction itself can be labeled so that other instructions can reference the constants symbolically.

```
ASSEMBLE(
    BR P + 1, I;
    CON LABELNAME );
ASSEMBLE (TAB: CON "ABCDEFGH"; .....
    LDB TAB, X; ..... );
```

Format 10

10a DMUL
DDIV
EADD
ESUB
EMPY
EDIV
ENEG
ECMP
DMPY

10b $\left\{ \begin{array}{l} \text{CVAD} \\ \text{CVBD} \end{array} \right\} \left[\begin{array}{l} 0 \\ 1 \end{array} \right]$

If item 2 is 0, 2 words are deleted from the stack.
If item 2 is 1 or empty, 4 words are deleted from the stack.

10c $\text{CVDB} \left[\begin{array}{l} 0 \\ 1 \end{array} \right]$

If item 2 is 0, 2 words are deleted from the stack.
If item 2 is 1 or empty, 3 words are deleted from the stack.

10d $\left\{ \begin{array}{l} \text{ADDD} \\ \text{SUBD} \\ \text{MPYD} \\ \text{CMPD} \\ \text{SLD} \\ \text{NSLD} \\ \text{SRD} \end{array} \right\} \left[\begin{array}{l} 0 \\ 1 \\ 2 \end{array} \right]$

If item 2 is 0, no words are deleted from the stack.
If item 2 is 1, 2 words are deleted from the stack.
If item 2 is 2 or empty, 4 words are deleted from the stack.

10e $\text{CVDA} \left[\begin{array}{l} 0 \\ 1 \\ \text{ABS} \\ \text{NABS} \end{array} \right] \left[\begin{array}{l} \left[\begin{array}{l} 0 \\ 1 \end{array} \right] \\ \left[\begin{array}{l} 0 \\ 1 \end{array} \right] \end{array} \right]$

If 0 is specified, 1 word is deleted from the stack.
If 1 is specified, 3 words are deleted from the stack.
If neither 0 nor 1 is specified, 3 words are deleted from the stack.
If ABS is specified, the target sign will be negative if the source is negative; otherwise, the target will be unsigned.
If NABS is specified, the target will be unsigned.
If neither ABS nor NABS is specified, the target sign will be the same as the source.

ADAX	2	BNOV	3a	DADD	2	DMUL	10a	FMPY	2
ADBX	2	BOV	3a	DASL	3b	DNEG	2	FNEG	2
ADD	2	BR	1c	DASR	3b	DPF	4b	FSUB	2
ADDD	10d	BRE	3a	DCMP	2	DSUB	2	HALT	6
ADDI	4a	BRO	3a	DCSL	3b	DTST	2	IABZ	3a
ADDM	1a	BTST	2	DCSR	3b	DUP	2	INCA	2
ADDS	7	CAB	2	DDEL	2	DXBZ	3a	INCB	2
ADXA	2	CIO	6	DDIV	10a	DXCH	2	INCM	1b
ADXB	2	CMD	6	DDUP	2	DZRO	2	INCX	2
ADXI	4a	CMP	2	DECA	2	EADD	10a	IXBZ	3a
AND	2	CMPB	2	DECB	2	ECMP	10a	✓IXIT	5
ANDI	7	CPMD	10d	DECM	1b	EDIV	10a	LADD	2
ASL	3b	CMPI	4a	DECX	2	EMPY	10a	LCMP	2
ASR	3b	CMPM	1a	DEL	2	ENEG	10a	LDB	1b
BCC	1d	CPMN	4a	DELB	2	ESUB	10a	LDD	1b
BCY	3a	CPRB	3a	DFLT	2	EXF	4b	✓LDEA	5
BE	1d	CSL	3b	DISP	6	EXIT	7	LDI	4a
BG	1d	CSR	3b	DIV	2	FADD	2	LDIV	2
BGE	1d	CVAD	10b	DIVI	4a	FCMP	2	LDNI	4a
BL	1d	CVBD	10b	DIVL	2	FDIV	2	LDPN	7
BLE	1d	CVDA	10e	✓DLSL	3b	FIXR	2	LDPP	7
BNE	1d	CVDB	10c	DLSR	3b	FIXT	2	LDX	1a
BNCY	3a	DABZ	3a	DMPY	10a	FLT	2	LDXA	2

LDXB	2	MTBA	1e	✓RCLK	6	STBX	2	XAX	2
LDXI	4a	MTBX	1e	RIO	6	STD	1b	XBZ	2
LDXN	4a	✓MTDS	8	RMSK	6	STOR	1a	XCH	2
LLBL	7	MVB	8a	RSW	5	SUB	2	XCHD	6
LLSH	5	MVBL	8c	SBXI	7	SUBD	10d	XEQ	6
LMPY	2	MVBW	8b	SCAL	7	SUBI	4a	XOR	2
LOAD	1a	MVLB	8c	SCAN	3b	SUBM	1a	XORI	7
✓LOCK	5	NEG	2	✓SCLK	6	SUBS	7	ZERO	2
LRA	1a	NOP	2	SCU	8c	SXIT	7	ZROB	2
✓LSEA	5	NOT	2	SCW	8c	TASL	3b	ZROX	2
LSL	3b	NSLD	10d	✓SDEA	5	TASR	3b		
LSR	3b	OR	2	SED	6	TBA	1e		
✓LST	6	ORI	7	SETR	4a	TBC	3b		
LSUB	2	PAUS	6	SIN	6	TBX	1e		
✓MABS	8	PCAL	7	SIO	6	TCBC	3b		
✓MDS	8	✓PCN	5	†SIRE	6	TEST	2		
✓MFDS	8	PLDA	5	SLD	10d	TIO	6		
MOVE	8a	✓PSDB	6	SMSK	6	TNSL	3b		
MPY	2	✓PSEB	6	SRD	10d	TRBC	3b		
MPYD	10d	PSHR	4a	✓SSEA	5	TSCB	3b		
MPYI	4a	PSTA	5	✓SST	6	†TSBM	3b		
MPYL	2	✓QASL	3b	STAX	2	✓UNLK	5	✓ - Series II only	
MPYM	1a	✓QASR	3b	STB	1b	WIO	6	† - Pre-Series II only	

00 STACK OPS

0	1	3	4	6	7	9	10	12	13	15
.	.	.	.	.	.	.	.	.	.	.

0 0

1st
POSITION2nd
POSITION

CC = condition, O = overflow, C = carry, Cl = clears bit

C	C	O	C		C	C	O	C	
				00 NOP					40 DEL
				01 DELB					41 ZROB
				02 DDEL	A				42 LDXB
A	X	X		03 ZROX	A				43 STAX
A	X	X		04 INCX	A				44 LDXA
A	X	X		05 DECX	A				45 DUP
				06 ZERO	A				46 DDUP
				07 DZRO	A				47 FLT
C				10 DCMF	C				50 FCMF
A	X	X		11 DADD	A	X			51 FADD
A	X	X		12 DSUB	A	X			52 FSUB
A	Cl	X		13 MPYL	A	X			53 FMPY
A	X			14 DIVL	A	X			54 FDIV
A	X			15 DNEG	A				55 FNEG
A				16 DXCH	A				56 CAB
C				17 CMP	C				57 LCMP
A	X	X		20 ADD	A		X		60 LADD
A	X	X		21 SUB	A		X		61 LSUB
A	X			22 MPY	A		X		62 LMPY
A	X			23 DIV	A	X			63 LDIV
A	X	X		24 NEG	A				64 NOT
A				25 TEST	A				65 OR
A				26 STBX	A				66 XOR
A		X		27 DTST	A				67 AND
A				30 DFLT	A	X	X		70 FIXR
B				31 BTST	A	X	X		71 FIXT
A				32 XCH					72 SPARE
A	X	X		33 INCA	A	X	X		73 INCB
A	X	X		34 DECA	A	X	X		74 DECB
A				35 XAX					75 XBX
A	X	X		36 ADAX	A	X	X		76 ADBX
A	X	X		37 ADXA	A	X	X		77 ADXB

01 SHIFTS/BRANCHES

0	1	3	4	6	7	9	10	12	13	15
.	.	.	.	.	.	.	.	.	.	.

A		0	1	X	0	0	0	← SC →	ASL
A				X	0	0	1	← SC →	ASR
A				X	0	0	2	← SC →	LSL
A				X	0	0	3	← SC →	LSR
A				X	0	0	4	← SC →	CSL
A				X	0	0	5	← SC →	CSR
A				X	0	0	6	0 0	SCAN
A	X	X		X	0	0	7	+/- ← P branch →	IABZ
A				X	0	1	0	← SC →	TASL
A				X	0	1	1	← SC →	TASR
A	X	X		1	0	1	2	+/- ← P branch →	IXBZ
A	X	X		1	0	1	3	+/- ← P branch →	DXBZ
		Cl		1	0	1	4	+/- ← P branch →	BCY
		Cl		1	0	1	5	+/- ← P branch →	BNCY
A				X	0	1	6	← SC →	TNSL
				1	7	0	0		SPARE
A				X	1	0	0	← SC →	DASL
A				X	1	0	1	← SC →	DASR
A				X	1	0	2	← SC →	DLSL
A				X	1	0	3	← SC →	DLSR
A				X	1	0	4	← SC →	DCSL
A				X	1	0	5	← SC →	DCSR
*				1	1	0	6	+/- ← P branch →	CPRB§
A	X	X		1	1	0	7	+/- ← P branch →	DABZ
		Cl		1	1	1	0	+/- ← P branch →	BOV
		Cl		1	1	1	1	+/- ← P branch →	BNOV
**				X	1	1	2	bit position	TBC
**				X	1	1	3	bit position	TRBC
**				X	1	1	4	bit position	TSBC
**				X	1	1	5	bit position	TCBC
				1	1	1	6	+/- ← P branch →	BRO
				1	1	1	7	+/- ← P branch →	BRE
C									
Cl									

Cl = clears bit

* = uses Index Reg.

** bit = 0 CCE, bit = 1 CCG or CCL

SC = shift count (0 - 63)

P branch signed magnitude (0 - 31)

bit position (0 - 63) [MOD 16]

CPRB X < (S-1) CCL, X > (5) CCG

(S-1) <= X <= (5) CCE

02 MOVES/IMMEDIATES

0	1-3	4-6	7-9	10-12	13-15
.	...	...	...	...	...

			0 2		
			0 0	0 PB DB 0	SDEC MOVE
			0 0	1 PB DB 0	SDEC MVB
			0 1	0	SDEC MVBL
B			0 1	1	SDEC MABS
			0 1	2	SDEC SCW
			0 1	3	SDEC MTDS
			0 1	4	SDEC MVLB
			0 1	5	SDEC MDS
			0 1	6	SDEC SCU
			0 1	7	SDEC MFDS
B			0 2	0 N A U	SDEC MVBW
C			0 2	1 PB DB 0	SDEC CMPB
A			0 3	0	0 RSW
A			0 3	0	1 LLSH§
A			0 3	2	0 PLDA§
			0 3	2	1 PSTA§
A			0 3	4	0 LSEA
			0 3	4	1 SSEA
A			0 3	4	2 LDEA
			0 3	4	3 SDEA
			0 3	6	0 IXIT
			0 3	6	1 LOCK
			0 3	6	2 PCN
			0 3	6	3 UNLK
A X			0 4	0	0 EADD
A X			0 4	0	1 ESUB
A X			0 4	0	2 EMPY
A X			0 4	0	3 EDIV
A			0 4	0	4 ENEG
C			0 4	0	5 ECOMP
			0 5	7	0 DMUL
			0 5	7	1 DDIV
A	X		0 6	0	1 DMPY
A X			0 6	0	2 CVAD
A X			0 6	0	3 CVDA
A X			0 6	0	4 CVBD

02 MOVES/IMMEDIATES (cont.)

0	1-3	4-6	7-9	10-12	13-15
.	...	...	...	...	...

A X			0	6	0	5	CVDB
A X X			0	6	0	6	SLD
A X X			0	6	0	7	NSLD
A X			0	6	1	0	SRD
A X			0	6	1	1	ADDD
C X			0	6	1	2	CMPD
A X			0	6	1	3	SUBD
A X			0	6	1	4	MPYD
A			1	0	← Imm Opr →		LDI
A			1	1	← Imm Opr →		LDXI
C			2	0	← Imm Opr →		CMPI
A X X			2	1	← Imm Opr →		ADDI
A X X			3	0	← Imm Opr →		SUBI
A X			3	1	← Imm Opr →		MPYI
A			4	0	← Imm Opr →		DIVI
			4	1	SBK DB DL Z STA X QS		PSHR
			5	0	← Imm Opr →		LDNI
			5	1	← Imm Opr →		LDXN
C			6	0	← Imm Opr →		CMPN
A			6	1 J J	J J K	K K K	EXF
A			7	0 J J	J J K	K K K	DPF
			7	1 SBK DB	DL Z STA	X QS	SETR
C	O	C					
C							

§ = uses Index Register

SDEC pop stack (0 - 3)

Imm Opr Immediate operand (0 - 255)

JJJJ Beginning bit position (0 - 15)

KKKK field length (0 - 15)

* SCW CARRY → Terminating character

* SCU CARRY → Terminating character

03 I/O, LINKAGE, CONTROL

0	1-3	4-6	7-9	10-12	13-15
...	...	...	...	...	...

			0	3	0	0	0	0	K	K	K	K	LST
					0	0	0	1	n	n	n	n	PAUS
					0	0	1	0	0	0	0	D/E	SED
3					0	0	6	0					XCHD
					0	0	6	1					PSDB
					0	0	6	2					DISP
3					0	0	6	3					PSEB
					0	1	0	0					SMSK
					0	1	0	1					SCLK
					0	1	2	0					RMSK
					0	1	2	1					RCLK
*					0	1	1	0	K	K	K	K	XEQ
D					0	1	1	1	K	K	K	K	SIO
D					0	2	0	0	K	K	K	K	RIO
D					0	2	0	1	K	K	K	K	WIO
1					0	2	1	0	K	K	K	K	TIO
1					0	2	1	1	K	K	K	K	CIO
					0	3	0	0	K	K	K	K	CMD
					0	3	0	1	K	K	K	K	SST
1					0	3	1	0	K	K	K	K	SIN
					0	3	1	1	n	n	n	n	HALT
					0	1	← STT →				SCAL		
					1	0	← STT →				PCAL		
					1	1	← SDEC + (4) →				EXIT		
					2	0	← SDEC + (1) →				SXIT		
A					2	1	← Imm Opr →				ADX1		
A					3	0	← Imm Opr →				SBXI		
					3	1	← PL - Disp →				LLBL		
					4	0	← P - Disp →				LDPP		
					4	1	← P - Disp →				LDPN		
					5	0	← Imm Opr →				ADDS		
					5	1	← Imm Opr →				SUBS		
					6	0	0	0	SPARE				
A					6	1	← Imm Opr →				ORI		
A					7	0	← Imm Opr →				XORI		
A					7	1	← Imm Opr →				ANDI		
C	O	C			KKKK Stack displacement (0 - 15)								
C					STT entry position (0 - 255)								
					Imm Opr Immediate operand (0 - 255) n Not used								

1. normally CCE, non-responding CCL
 D norm CCE, non-respond CCL, not ready CGG
 3. norm CCE, IF error CCL
 • XEQ depends on instruction executed.

MEMORY REFERENCE

0	1-3	4-6	7-9	10-12	13-15
...	...	...	...	...	...

A			0	4	X	1	0			P			LOAD
A			0	5	X	1	1			DQS			
			0		0					+/-			P rel branch → TBA
					2					+/-			P rel branch → MTBA
					4					+/-			P rel branch → TBX
					6					+/-			P rel branch → MTBX
					X	1	1			DQS			STOR
C			0	6	X	1	0			P			CMPM
C					X	1	1			DQS			
A	X	X	0	7	X	1	0			P			ADDM
A	X	X			X	1	1			DQS			
A	X	X	1	0	X	1	0			P			SUBM
A	X	X			X	1	1			DQS			
A	X	X	1	1	X	1	0			P			MPYM
A	X	X			X	1	1			DQS			
A	X	X	1	2	X	1	0			DQS			INCM
A	X	X			X	1	1			DQS			DECM
A			1	3	X	1	0			P			LDX
A					X	1	1			DQS			
			1	4	X	1	0			+/-			P rel branch → BR
					X	1	1			DQS indirect			BR
					I	0	1			GEL	+/-		P branch → BCC
B			1	5	X	1	0			DQS			LDB
A					X	1	1			DQS			LDD
			1	6	X	1	0			DQS			STB
					X	1	1			DQS			STD
			1	7	X	1	0			P			LRA
					X	1	1			DQS			
C	O	C											

P	P+	0								0:377
	P-	1								0:377
	DB+	0								0:377
	Q+	1	0							0:177
DQS	Q-	1	1	0						0:77
	S-	1	1	1						0:77

- GEL { 1. Less than (BL) 4. Greater than (BG)
 2. Equal (BE) 5. Not equal (BNE)
 3. Less than or equal (BLE) 6. Greater or equal (BGE)

STATUS REGISTER

0	1	3	4	6	7	9	10	12	13	15
M	I	T	R	O	C	CC	SEGMENT NUMBER			

J K

M Mode 1 = Priv.

I Ext Interrupts

T User Traps

R Left/Right stack op

CC field

6-7 Condition

0	> G
1	< L
2	= E
3	Undef.

CONDITION CODES

K (bits (7:9))

0	1	2	3	4	5	6	7	O	C
0	G	G	G	G	L	L	L	L	0
1	E	E	E	E	L	L	L	L	0
2	G	G	G	G	L	L	L	L	0
3	E	E	E	E	L	L	L	L	0
4	G	G	G	G	L	L	L	L	0
5	E	E	E	E	L	L	L	L	0
6	G	G	G	G	L	L	L	L	0
7	E	E	E	E	L	L	L	L	0

OVERFLOW
CARRY

CCA

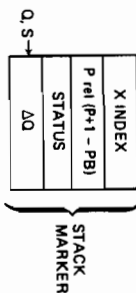
opr < 0
opr = 0
opr > 0

CCB

spec ASCII
Alphabetic
Numeric

CCC

opr 1 < opr 2
opr 1 = opr 2
opr 1 > opr 2



```

:SPL [textfile] [, [usfile] [, [listfile] [, [masterfile] [, newfile] ] ] ]
Default: $STDIN $NEWPASS $STDLIST $NULL $NULL

:SPLPREP [textfile] [, [progrfile] [, [listfile] [, [masterfile] [, newfile] ] ] ]
Default: $STDIN $NEWPASS $STDLIST $NULL $NULL

:SPLGO [textfile] [, [listfile] [, [masterfile] [, newfile] ] ]
Default: $STDIN $STDLIST $NULL $NULL (Progrfile is still in $OLDPASS after :SPLGO.)

:EDITOR [listfile]

:PREP usfile , progrfile {;ZERODB} {;PMAP} {;MAXDATA = segsize} {;STACK = stacksize} {;DL = dlsiz}
Set DL-DB and List proc. Max size DL-Z. Max size Qi-Z. Initial DL-DB area.
S-Z to all zero. map.

[;CAP = caplist] (Capabilities = IA, BA, PH, DS, MR, RT, CR, PM, NS.)

:RUN progrfile [, entrypoint] {;NOPRIV} {;LMAP} {;MAXDATA = segsize} {;PARAM = paramnum} {;STACK = stacksize}
Override priv List alloc Max size DL-Z. Passed in Q+4 [;DEBUG]
mode setting. of proc. Initial DL-DB area.

[;DL = dlsiz] [;LIB = library] (Order of library search: G = logon group first, P = public group first, S = public group of system
account first.)

```

NOTES: There must be a space after the command name and before first parameter. Commands must not use columns 73-80, but can be continued with ampersand (&).

\$PAGE ["character string" list] (LIST must be on. \$PAGE not listed itself. String becomes TITLE.)

\$TITLE ["character string" list] (Uses character string as running title on each page. Default = null.)

\$TRACE [program-unit] : identifier, ...

\$SET [Xn = $\begin{Bmatrix} \text{ON} \\ \text{OFF} \end{Bmatrix}$] [, Xn = $\begin{Bmatrix} \text{ON} \\ \text{OFF} \end{Bmatrix}$] ... (n between 0 and 9. Sets toggle n ON or OFF. Default is all toggles set to OFF.)

\$IF [Xn = $\begin{Bmatrix} \text{ON} \\ \text{OFF} \end{Bmatrix}$] (If condition false, source records are ignored until another IF.)

\$EDIT param [, param] ... (Masterfile and textfile are merged by sequence number—columns 73 to 80—to newfile. Textfile record is taken if numbers match. If no masterfile, then no sequence checking.)

Params: NOSEQ (stop renumbering)

VOID = $\begin{Bmatrix} \text{number} \\ \text{"string"} \end{Bmatrix}$ (left-fill with "0") (Skips masterfile up to record = sequence number specified.)

SEQNUM = number (Renums merged source records starting at number. Sets INC = 1000.)

INC = number (Default: 1000) (Sets increment for renumbering; SEQNUM must also be specified.)

\$CONTROL [param, ...]

Params: LIST, NOLIST, WARN, NOWARN, MAP, ERRORS = nnn, CODE, NOCODE, SEGMENT = segname, MAIN = proname, USLINIT, LINES = nnn, ADR, INNERLIST, UNCALLABLE, PRIVILEGED, SUBPROGRAM {[proc[""] [, proc[""]]] ...}

Default: LIST, WARN, MAP off, ERRORS = 100, NOCODE, SEGMENT = SEG, MAIN = OB, ADR off, INNERLIST off, LINES = 60, USL uninitialized unless compiler opens a new temporary USL file (i.e. \$NEWPASS) program mode (not subprogram).

Note: NOLIST turns off INNERLIST, ADR. SUBPROGRAM compiles only the procedures listed (unless none listed, then all are compiled). Those with * are automatically compiled with LIST, MAP and CODE.

These symbols are reserved and cannot be identifiers:

ABSOLUTE	ELSE	LABEL	PUSH
ALPHA	END	LAND	REAL
AND	ENTRY	LOGICAL	RETURN
ARRAY	EQUATE	LONG	SCAN
ASSEMBLE	EXTERNAL	LOR	SET
BEGIN	FALSE	MOD	SINGLE
BYTE	FIXR	MODD	SPECIAL
CARRY	FIXT	MOVE	STEP
CASE	FOR	NOCARRY	SUBROUTINE
CAT	FORWARD	NOT	SWITCH
CHECK	GLOBAL	NOVERFLOW	THEN
COMMENT	GO	NUMERIC	TO
DABZ	GOTO	OF	TOS
DDEL	IABZ	OPTION	TRUE
DEFINE	IF	OR	UNCALLABLE
DEL	INTEGER	OVERFLOW	UNTIL
DELB	INTERNAL	OWN	VALUE
DO	INTERRUPT	POINTER	VARIABLE
DOUBLE	INTRINSIC	PRIVILEGED	WHILE
DXBZ	IXBZ	PROCEDURE	XOR

THE INDEX REGISTER

As Variable: $\begin{Bmatrix} \text{BYTE} \\ \text{LOGICAL} \\ \text{INTEGER} \end{Bmatrix}$ identifier = X; X not reserved.

Where used in SPL/3000:

- Whenever variable as above is used.
- Indexing arrays or pointers: $\begin{Bmatrix} \text{array} \\ \text{pointer} \end{Bmatrix}$ (index)
- Variable shift count in X: primary & shiftopt (count)
- int expr <= int expr <= int expr (CPRB instr)
- CASE and Indexed GOTO statement (i.e., SWITCH).
- IXBZ and DXBZ in Condition Clause.
- ABSOLUTE (address) as variable. (PLDA, PSTA inst)
- During allocation of local variables in procedures with multiple entry points.

HEWLETT  PACKARD

Part No. 32100-90001
Product No. 32100A

Printed in U.S.A.

NOV 1976